

Министерство науки и высшего образования Российской Федерации  
федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«Кузбасский государственный технический университет  
имени Т. Ф. Горбачева»

Институт профессионального образования  
Кафедра информатики и информационных систем

Юлия Сергеевна Дементьева

## **ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЯ**

Методические материалы к практическим занятиям и  
самостоятельной работе

Рекомендовано цикловой методической комиссией  
специальности 09.02.07 Информационные системы и  
программирования в качестве электронного издания для  
использования в образовательном процессе

Кемерово 2024

Рецензенты: Семенова О.С. – кандидат тех. наук, заведующей кафедрой информатики и информационных систем ИПО ФГБОУ ВО «Кузбасский государственный технический университет имени Т. Ф. Горбачева».

**Дементьева, Ю.С. Основы алгоритмизации и программирования:** методические материалы к практическим занятиям и самостоятельной работе для обучающихся направления подготовки (специальности) 09.02.07 специализации «Информационные системы и программирование» всех форм обучения / сост. Ю. С. Дементьева; Кузбасский государственный технический университет имени Т. Ф. Горбачева. – Кемерово, 2024. – Текст: электронный.

Приведено содержание практических работ, порядок их оформления, а также материал, необходимый для успешного изучения дисциплины. Назначение издания – помощь обучающимся в получении знаний по дисциплине «Основы алгоритмизации и программирования» и организация практических работ.

© Кузбасский  
государственный технический  
университет имени Т. Ф.  
Горбачева, 2024  
© Дементьева Ю.С.,  
составление, 2024

## ОГЛАВЛЕНИЕ

ПРАКТИЧЕСКАЯ РАБОТА №1. АЛГОРИТМИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА .....	4
ПРАКТИЧЕСКАЯ РАБОТА №2. ИЗУЧЕНИЕ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТЧИКА .....	11
ПРАКТИЧЕСКАЯ РАБОТА №3. ИЗУЧЕНИЕ ОСНОВНЫХ ОБЪЕКТОВ И ИХ СВОЙСТВ .....	17
ПРАКТИЧЕСКАЯ РАБОТА №4. СВОЙСТВА И МЕТОДЫ ОБЪЕКТОВ. РЕАКЦИЯ ОБЪЕКТОВ НА СОБЫТИЯ.....	33
ПРАКТИЧЕСКАЯ РАБОТА №5. ПОСТРОЕНИЕ ВЫРАЖЕНИЙ .....	38
ПРАКТИЧЕСКАЯ РАБОТА №6. СТРОКИ .....	50
ПРАКТИЧЕСКАЯ РАБОТА №7. РАБОТА С МАССИВАМИ .....	60
ПРАКТИЧЕСКАЯ РАБОТА №8. ПРОГРАММИРОВАНИЕ МОДУЛЯ. ОРГАНИЗАЦИЯ ПРОЦЕДУР И ФУНКЦИЙ.....	68
ПРАКТИЧЕСКАЯ РАБОТА №9. ОРГАНИЗАЦИЯ ДОСТУПА К ОБЪЕКТАМ ТАБЛИЧНОГО ПРОЦЕССОРА .....	74
ПРАКТИЧЕСКАЯ РАБОТА №10. АВТОМАТИЗАЦИЯ СТАНДАРТНЫХ ТЕКСТОВЫХ ДОКУМЕНТОВ .....	81
ПРАКТИЧЕСКАЯ РАБОТА №11. ОРГАНИЗАЦИЯ ДОСТУПА К ОБЪЕКТАМ БАЗЫ ДАННЫХ .....	91
ПРАКТИЧЕСКАЯ РАБОТА №12. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ.....	96
САМОСТОЯТЕЛЬНАЯ РАБОТА №1 .....	102
САМОСТОЯТЕЛЬНАЯ РАБОТА №2 .....	106
САМОСТОЯТЕЛЬНАЯ РАБОТА №3 .....	110
САМОСТОЯТЕЛЬНАЯ РАБОТА №4 .....	115
САМОСТОЯТЕЛЬНАЯ РАБОТА №5 .....	118
САМОСТОЯТЕЛЬНАЯ РАБОТА №6 .....	122
Литература .....	126

## **ПРАКТИЧЕСКАЯ РАБОТА №1. АЛГОРИТМИЗАЦИЯ ВЫЧИСЛИТЕЛЬНОГО ПРОЦЕССА**

**Цель работы:** усвоить понятия алгоритма как фундаментальное понятие информатики, способы описания, основные типы алгоритмов. Освоить принципы решения задач с использованием основных алгоритмических конструкций.

### **ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ**

Решение любой задачи на ЭВМ можно разбить на следующие этапы: разработка алгоритма решения задачи, составление программы решения задачи на алгоритмическом языке, ввод программы в ЭВМ, отладка программы (исправление ошибок), выполнение программы на ПК, анализ полученных результатов.

**Алгоритм** – это четкая последовательность действий, выполнение которой дает какой-то заранее известный результат. Простыми словами, это набор инструкций для конкретной задачи. Известнее всего этот термин в информатике и компьютерных науках, где под ним понимают инструкции для решения задачи эффективным способом.

### **Способы описания алгоритмов**

На любой стадии существования алгоритмы представляют с помощью конкретных изобразительных средств. Состав и правила употребления которых образуют конкретные способы или формы записи. К настоящему времени сложились пять наиболее употребительных способов записи:

- словесное описание;
- формульно-словесное описание;
- псевдокод;
- графический способ (блок-схема);
- программа (способ описания с помощью языков программирования).

### **Основные виды алгоритмов**

Несмотря на слово «последовательность», алгоритм не всегда описывает действия в жестко заданном порядке. Особенно

это актуально сейчас, с распространением асинхронности в программировании. В алгоритмах есть место для условий, циклов и других нелинейных конструкций.

**Линейные.** Это самый простой тип алгоритма: действия идут друг за другом, каждое начинается после того, как закончится предыдущее. Они не переставляются местами, не повторяются, выполняются при любых условиях.

**Ветвящиеся.** В этом типе алгоритма появляется ветвление: какие-то действия выполняются, только если верны некоторые условия. Например, если число меньше нуля, то его нужно удалить из структуры данных. Можно добавлять и вторую ветку: что делать, если условие неверно – например, число больше нуля или равно ему. Условий может быть несколько, они могут комбинироваться друг с другом.

**Циклические.** Такие алгоритмы выполняются в цикле. Когда какой-то блок действий заканчивается, эти действия начинаются снова и повторяются некоторое количество раз. Цикл может включать в себя одно действие или последовательность, а количество повторений может быть фиксированным или зависеть от условия: например, повторять этот блок кода, пока в структуре данных не останется пустых ячеек. В некоторых случаях цикл может быть бесконечным.

**Рекурсивные.** Рекурсия – это явление, когда какой-то алгоритм вызывает сам себя, но с другими входными данными. Это не цикл: данные другие, но «экземпляров» работающих программ несколько, а не одна. Известный пример рекурсивного алгоритма – расчет чисел Фибоначчи.

Рекурсия позволяет изящно решать некоторые задачи, но с ней надо быть осторожнее: такие алгоритмы могут сильно нагружать ресурсы системы и работать медленнее других.

**Вероятностные.** Такие алгоритмы упоминаются реже, но это довольно интересный тип: работа алгоритма зависит не только от входных данных, но и от случайных величин. К ним, например, относятся известные алгоритмы Лас-Вегас и Монте-Карло.

**Основные и вспомогательные.** Это еще один вид классификации. Основной алгоритм решает непосредственную

задачу, вспомогательный решает подзадачу и может использоваться внутри основного – для этого там просто указываются его название и входные данные. Пример вспомогательного алгоритма – любая программная функция.

### **Основные свойства алгоритмов**

- Дискретность (прерывность) – разбиение алгоритма на шаги;
- Результативность – получение из исходных данных результата за конечное число шагов;
- Массовость – пригодность для решения не какой-либо одной, а целого класса задач;
- Детерминированность (определенность) – совпадение получаемых результатов независимо от пользователя и применяемых технических средств;
- Выполнимость и понятность – каждый шаг алгоритма должен быть понятен исполнителю.

### **Графический способ записи алгоритмов**

**Блок-схема** – это схематичное представление процесса, системы или компьютерного алгоритма. Блок-схемы часто применяются в разных сферах деятельности, чтобы документировать, изучать, планировать, совершенствовать и объяснять сложные процессы с помощью простых логичных диаграмм. Для построения блок-схем применяются прямоугольники, овалы, ромбы и некоторые другие фигуры (для обозначения конкретных операций), а также соединительные стрелки, которые указывают последовательность шагов или направление процесса.



Блок-схемы варьируются от незамысловатых, нарисованных вручную до подробных, составленных на компьютере диаграмм со множеством шагов и процессов. Если учесть все возможные вариации, блок-схемы можно признать одним из самых распространенных видов схем во всем мире. Они широко используются в разных сферах как технической, так и нетехнической направленности. Иногда блок-схемы получают более узкоспециальные названия, например, схема процесса, схема рабочего процесса, функциональная блок-схема,

моделирование бизнес-процессов, модель и нотация бизнес-процессов (BPMN) или схема технологического процесса (PFD).

Такое графическое представление называется схемой алгоритма или блок-схемой. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий. В таблице 1.1 приведены наиболее часто употребляемые символы.

Таблица 1.1 – Элементы блок-схем

Название символа	Обозначение и пример заполнения	Пояснение
Процесс		Вычислительное действие или последовательность действий
Решение		Проверка условий
Модификация		Начало цикла
Предопределенный процесс		Вычисления по подпрограмме, стандартной подпрограмме
Ввод-вывод		Ввод-вывод в общем виде
Пуск-останов		Начало, конец алгоритма, вход и выход в подпрограмму

Документ		Вывод результатов на печать
Комментарий		В сочетании с другими материалами этот символ позволяет добавить необходимый контекст, разъяснение или комментарий к определенному диапазону данных. Комментарий также можно присоединить к необходимому разделу блок-схемы с помощью пунктирной линии.

**Блок «процесс»** применяется для обозначения действия или последовательности действий, изменяющих значение, форму представления или размещения данных. Для улучшения наглядности схемы несколько отдельных блоков обработки можно объединять в один блок. Представление отдельных операций достаточно свободно.

**Блок «решение»** используется для обозначения переходов управления по условию. В каждом блоке «решение» должны быть указаны вопрос, условие или сравнение, которые он определяет.

**Блок «модификация»** используется для организации циклических конструкций. (Слово модификация означает видоизменение, преобразование). Внутри блока записывается параметр цикла, для которого указываются его начальное значение, граничное условие и шаг изменения значения параметра для каждого повторения.

**Блок «предопределенный процесс»** используется для указания обращений к вспомогательным алгоритмам,



существующим автономно в виде некоторых самостоятельных модулей, и для обращений к библиотечным подпрограммам.

**Блок «Ввод-вывод»** используется для преобразования данных в форму, пригодную для обработки (ввод) или отображения результатов обработки (вывод). Отдельным логическим устройствам компьютера или отдельным функциям обмена соответствуют определенные блочные символы. В каждом из них указываются тип устройства или файла данных, тип информации, участвующий в обмене, а также вид операции обмена.

**Блок «Пуск-останов»** используется для обозначения начала, конца, прерывания процесса обработки данных или выполнения программы.

**Блок «Документ»** предназначен для ввода-вывода данных, носителем которых служит бумага.

**Пример 1.** Рассмотрим алгоритм «Сложение 2-х чисел» и изобразим его в виде блок-схемы (рис.1.1).

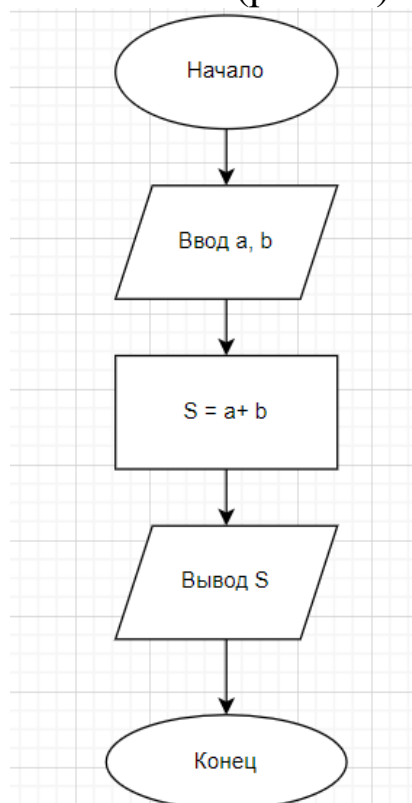


Рисунок 1.1 – Блок схема алгоритма «Сложение 2-х чисел»

## **ЗАДАНИЯ К ПРАКТИЧЕСКОЙ РАБОТЕ №1**

**Задание 1.** Разработать, в соответствии с вариантом, алгоритм линейной структуры и представить его в виде блок-схемы.

1. Найти площадь прямоугольника. Данные: длина и ширина.
2. Рассчитать периметр квадрата. Данные: длина стороны.
3. Определить площадь треугольника. Данные: база и высота.
4. Найти периметр правильного треугольника. Данные: длина стороны.
5. Рассчитать площадь круга. Данные: радиус.
6. Определить длину окружности. Данные: радиус.
7. Найти площадь параллелограмма. Данные: база и высота.
8. Рассчитать объем куба. Данные: длина ребра.
9. Определить объем призмы. Данные: площадь основания и высота.
10. Найти площадь трапеции. Данные: длина верхнего основания, длина нижнего основания и высота.

**Задание 2.** Разработать, в соответствии с вариантом, алгоритм разветвленной структуры и представить его в виде блок-схемы.

1. Даны 2 числа. Определить, являются числа четными или нечетными.
2. Дано число. Определить, является число положительным, отрицательным или нулевым.
3. Даны 3 числа. Определите количество нечетных чисел.
4. Даны 2 числа. Определить количество положительных чисел.
5. Даны 4 числа. Определить сколько чисел из них равны нулю.
6. Дано трехзначное число. Определить является ли оно простым.
7. Даны 3 числа. Определить какое из 3-х чисел является наибольшим.
8. Даны 2 числа. Определить количество чисел кратных 2.
9. Даны 3 числа. Определить количество отрицательных чисел.
10. Дано число. Определить является ли число степенью двойки.

## ПРАКТИЧЕСКАЯ РАБОТА №2. ИЗУЧЕНИЕ ИНТЕГРИРОВАННОЙ СРЕДЫ РАЗРАБОТЧИКА

**Цель работы:** ознакомиться со средой разработчика программ Visual Studio 2022, изучить основные окна и написать консольное приложение.

### ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

**Консольное приложение** – программа, работающая в режиме текстового интерфейса.

Алгоритм создания консольного приложения:

1. Откройте Visual Studio 2019
2. В появившемся диалоговом окне, выбираем Создание проекта (рис 2.1)

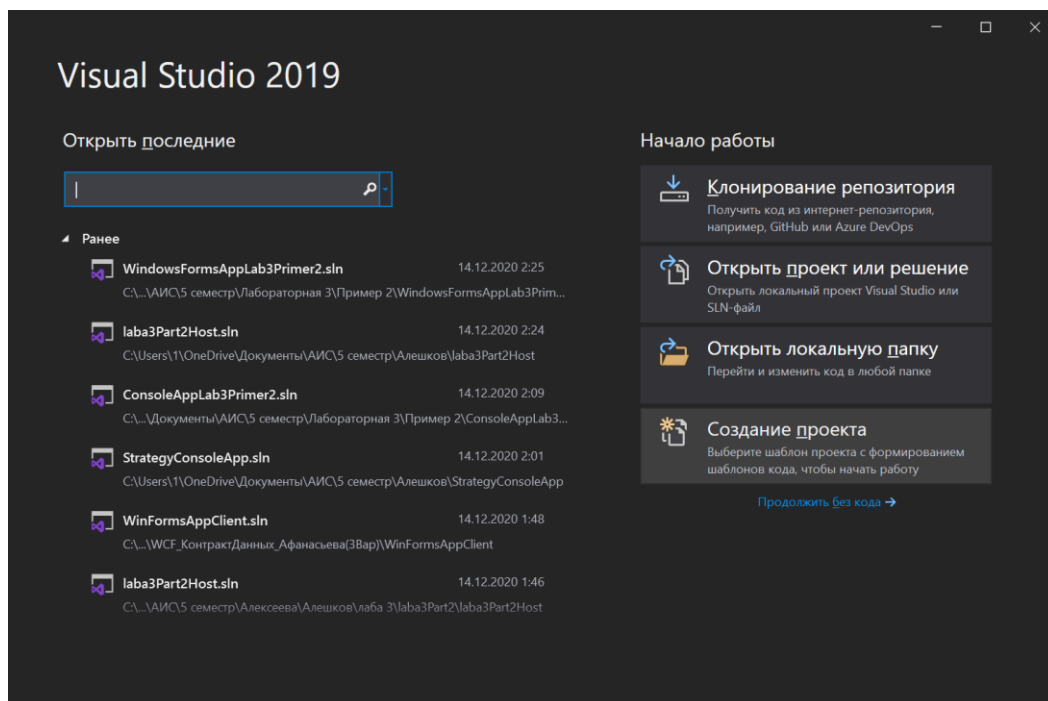


Рисунок 2.1 – Создание проекта

3. В окне Создание проекта выбираем «Консольное приложение (.NET Framework) C#» и нажимаем и на кнопку «Далее» (рис. 2.2).

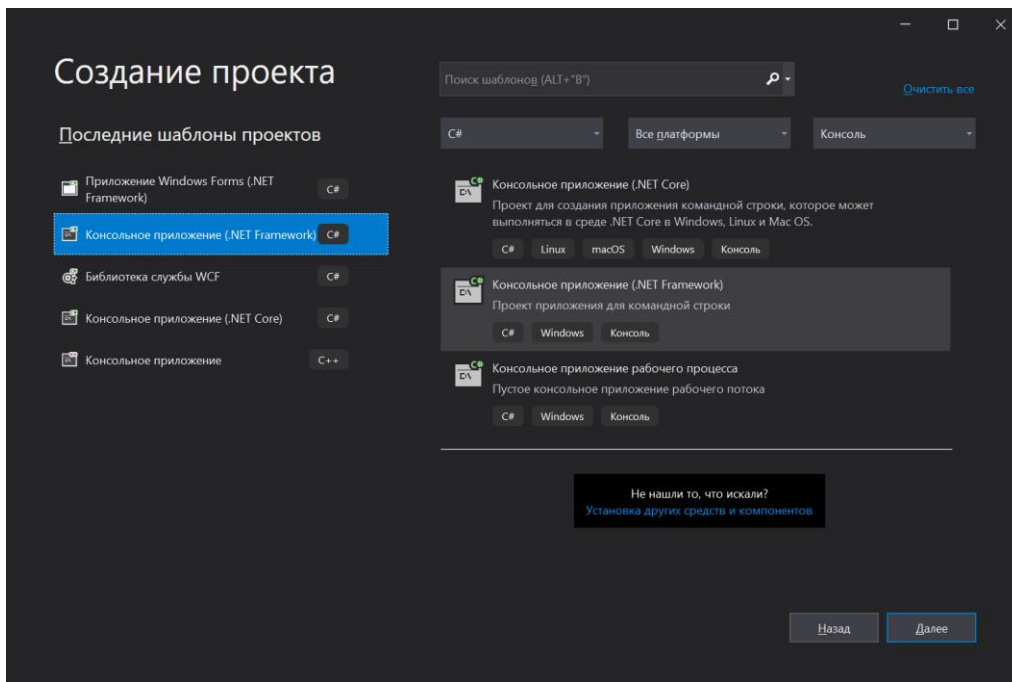


Рисунок 2.2 – Создание проекта

4. В следующем окне задаём имя проекта и его расположение. Затем нажимаем кнопку Создать (рис. 2.3). Обязательно измените имя проекта и выберите место для его хранения, таким образом, чтобы вы точно знали где проект будет у Вас сохраняться.

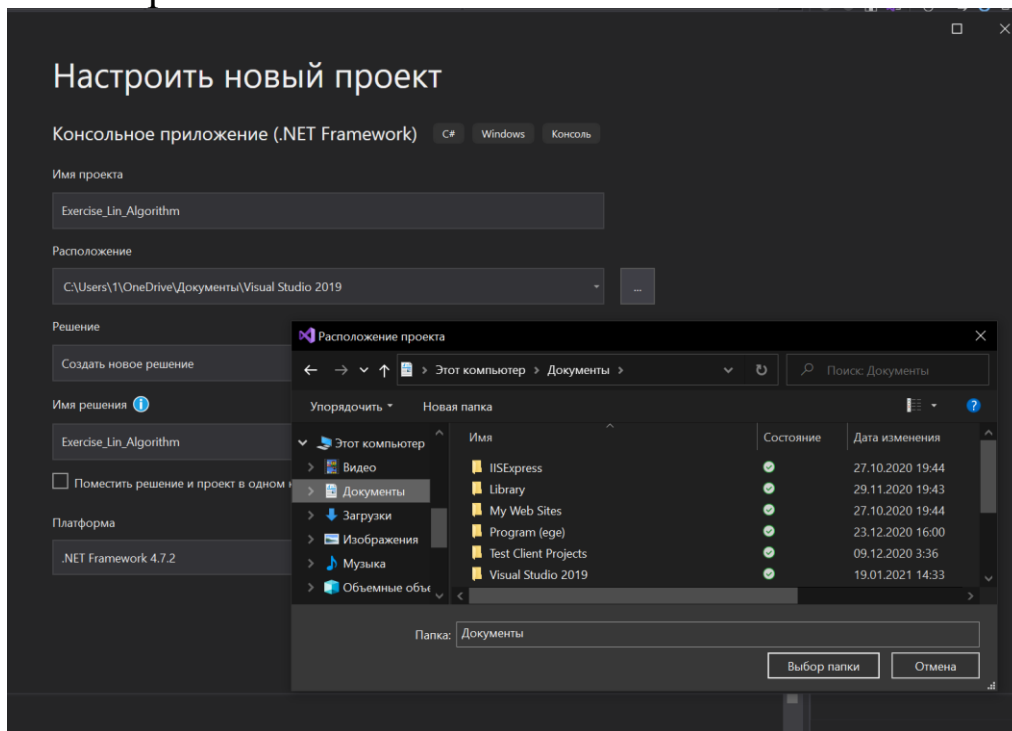


Рисунок 2.3 – Создание проекта

5. Внешний вид открытого консольного приложения в Visual Studio выглядит следующим образом (рис. 2.4).

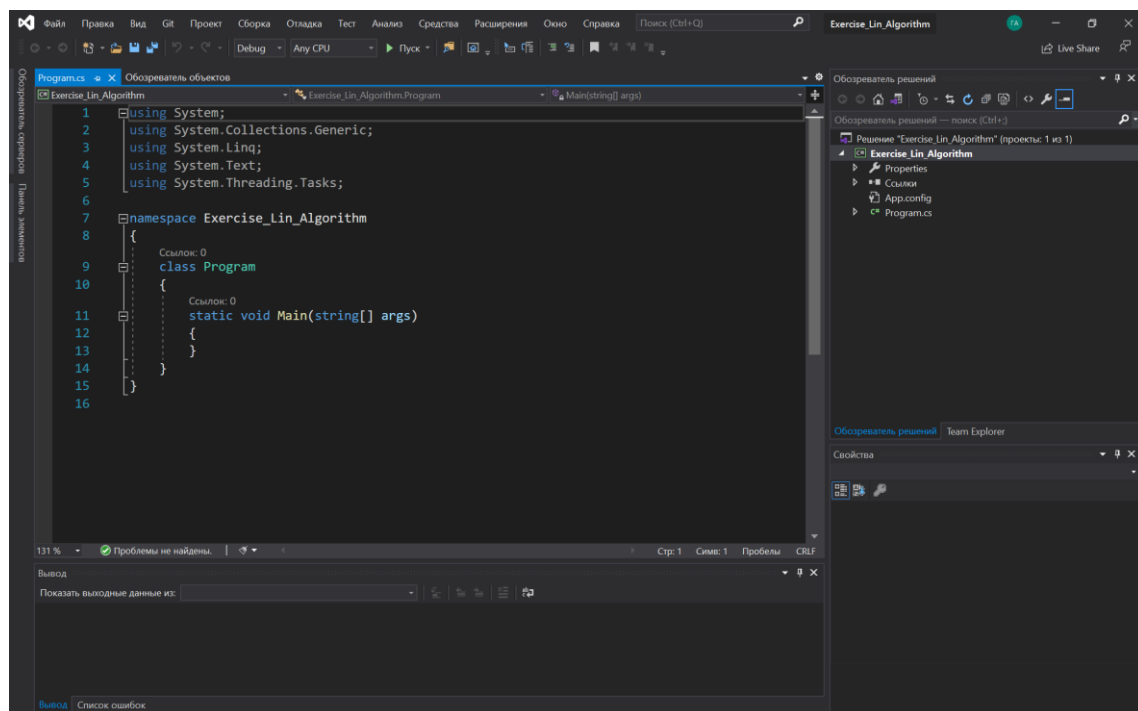


Рисунок 2.4 – Создание проекта

6. Программный код следует располагать в файле Program.cs (открывается автоматически) внутри метода Main:

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace Exercise_Lin_Algorithm
{
    class Program
    {
        static void Main(string[] args)
        {
            // ВАШ ПРОГРАММНЫЙ КОД
        }
    }
}
```

## ПРИМЕР ВЫПОЛНЕНИЯ ЗАДАНИЯ

Напишите программу для решения следующей задачи:

1. Вычислить площадь кольца и длины окружностей кольца (внешней и внутренней), если известно, что радиус внутренней окружности кольца равен  $R$ , а радиус внешней окружности в два раза больше внутренней.

Решение задачи находится ниже (рис 2.5 – 2.7):

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

```

### Рисунок 2.5 - Решение

```

namespace Exercise_Lin_Algorithm
{
    class Program
    {
        static void Main(string[] args)
        {
            // Описание переменных
            double R;
            double S, L1, L2;

            // Ввод исходных данных
            Console.Write("Введите значение внутреннего радиуса кольца
");

            R = Convert.ToDouble(Console.ReadLine());

            // Вычисления
            S = Math.PI * (Math.Pow(2 * R, 2) - Math.Pow(R, 2));
            L1 = 4 * Math.PI * R;
            L2 = 2 * Math.PI * R;

            // Вывод результата
            Console.WriteLine("Площадь кольца равна "+ S);

```

### Рисунок 2.6 - Решение

```

            Console.WriteLine("Длина внешней окружности равна {0}, а
длина внутренней окружности - {1}", L1, L2);
            Console.ReadKey();
        }
    }
}

```

### Рисунок 2.7 - Решение

Рассмотрим данный код подробнее:

1. В конце каждой строки программы ставится символ «;»
2. `double R` – объявляется переменная `R` тип данных `double` (вещественный, может принимать целые и дробные значения)
3. `int R` – объявляется переменная целого типа
4. `Console.Write()` и `Console.WriteLine()` – команды вывода данных в консоль
5. `Console.WriteLine()` – выводит данные записанные в скобках в консоль и переводит курсор на следующую строку
6. `Console.Write()` – выводит данные записанные в скобках в консоль при этом курсор остаётся на текущей строке
7. `Console.ReadLine()` и `Console.Read()` – команды считывания данных введённых пользователем в консоль  
Например, `str = Console.ReadLine();`
8. Данные, введенные пользователем, помещаются в переменную `str`. Переменная `str` строкового типа. Все данные, записываемые пользователем в консоле только строковые. Следовательно, их необходимо преобразовать к нужному типу данных.
9. `R=Convert.ToDouble(str)` – данные находящиеся в переменной `str` преобразуются к типу данных `double` и помещаются в переменную `R`
10. При описании переменных `str` задаётся как строка (`string str;`), а `R` – как вещественное число (`double R;`)
11. `Convert.ToInt32()` – преобразование в целочисленный тип
12. `Math.PI` – число  $\pi$   
`Math.Pow(x, y)` – возведение `x` в степень `y` (`xy`)
13. Вывод данных осуществляется с помощью строки:  
`Console.WriteLine("Площадь кольца равна "+ S);`
14. `+` является знаком конкатенации (соединения строк)  
`Console.WriteLine("Длина внешней окружности равна {0}, а длина внутренней окружности - {1}",L1,L2);`
15. При выводе на экран переменная `L1` помещается вместо `{0}`, а `L2` – вместо `{1}`

**ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №2.**

Разработайте консольное приложение. Реализуйте алгоритмы из практической работы №1 на языке C# в соответствии со своим вариантом.



## **ПРАКТИЧЕСКАЯ РАБОТА №3. ИЗУЧЕНИЕ ОСНОВНЫХ ОБЪЕКТОВ И ИХ СВОЙСТВ**

**Цель работы:** изучить основные элементы управления Windows -форм, их свойства и методы, а также получить практические навыки в разработке Windows - форм с элементами контроля.

### **ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ**

Для Windows-приложений, ориентированных на выполнение в общезыковой исполняющей среде (CLR), в качестве основы построения графического пользовательского интерфейса используются классы Windows Forms из библиотек .NET Framework, т.е. Windows Forms – это набор средств для создания Windows-приложений, выполняющихся в среде CLR.

Средства Windows Forms обеспечивают быструю разработку пользовательского интерфейса, поскольку он создается из стандартных элементов, а соответствующий программный код генерируется автоматически. Затем требуется лишь доработать сгенерированный код, чтобы добиться необходимой функциональности.

Применяя Windows Forms, можно построить удобный пользовательский интерфейс, в том числе главное окно приложения, выбирая подходящие элементы управления, с которыми взаимодействует пользователь.

Чтобы создать Windows-приложение на основе Windows Forms, следует выбрать команду главного меню «Файл» (File) → «Создать» (New) → «Проект» (Project), затем выбрать тип проекта «Рабочий стол» (Windows Desktop), а в качестве шаблона проекта указать Windows Forms Application (Рисунок 3.1).

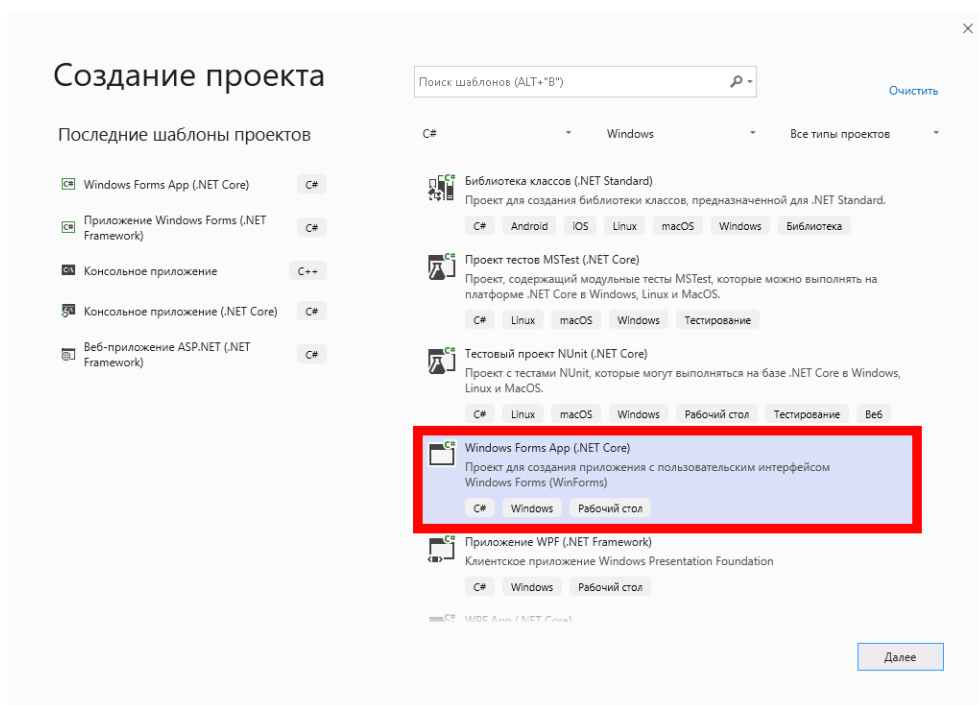


Рисунок 3.1 – Создание приложения

После этого следует ввести имя проекта, например, PR1, и указать папку для хранения проекта.

После этого Visual Studio откроет наш проект с созданными по умолчанию файлами:

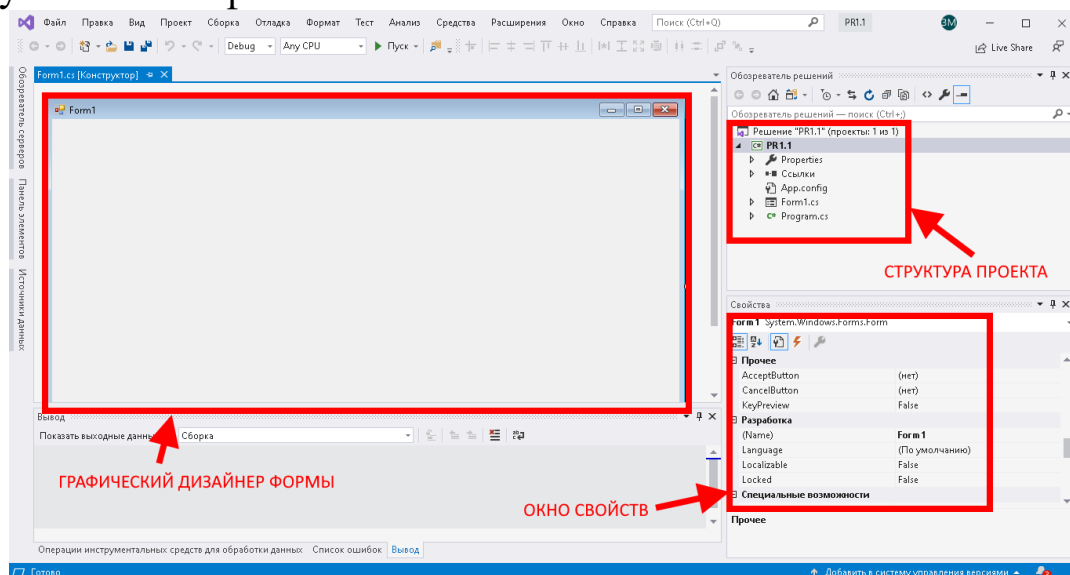


Рисунок 3.2 – Окно проекта

Большую часть пространства Visual Studio занимает графический дизайнер, который содержит форму будущего приложения. Пока она пуста и имеет только заголовок Form1. Справа находится окно файлов решения/проекта - Solution Explorer (Обозреватель решений). Там и находятся все связанные

с нашим приложением файлы, в том числе файлы формы Form1.cs.

Внизу справа находится окно свойств - Properties. Так как в данный момент выбрана форма как элемент управления, то в этом поле отображаются свойства, связанные с формой.

### Размещения элементов управления на форме

Для размещения различных элементов управления на форме используется Панель инструментов (Toolbox). Панель элементов содержит элементы управления, сгруппированные по типу. Каждую группу элементов можно свернуть, если она в настоящий момент не нужна. (Рисунок 3.3)

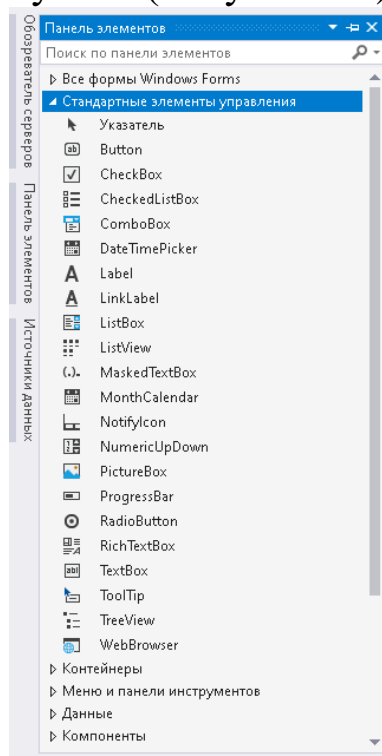


Рисунок 3.3 – Панель элементов

Чтобы добавить элементы управления на форму, необходимо щелкнуть на добавляемый элемент в панели инструментов, а затем щелкнуть в нужном месте формы. После этого элемент появится на форме (Рисунок 4).

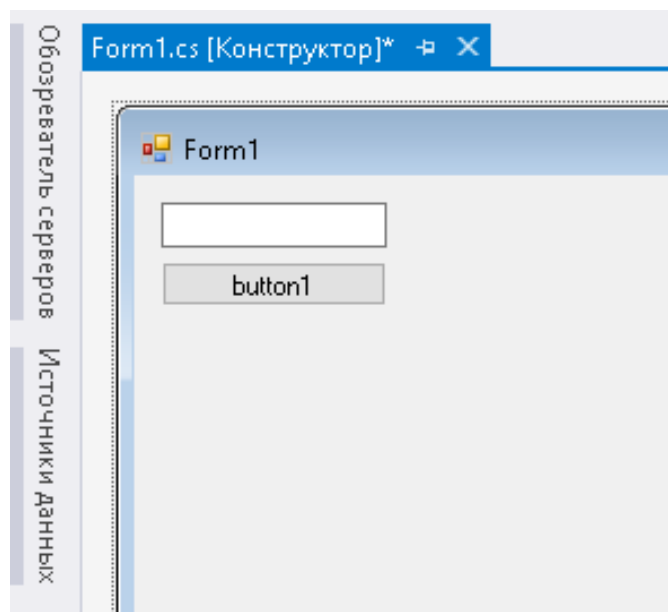


Рисунок 3.1 – Форма с добавленными элементами управления (TextBox и Button)

Элемент можно перемещать по форме, схватившись за него левой кнопкой мыши. Если элемент управления позволяет изменять размеры, то на соответствующих его сторонах появятся белые кружки, ухватившись за которые и можно изменить размер. При размещении элемента управления на форме, его можно выделить щелчком мыши и получить доступ к его свойствам в окне свойств.

### **Размещение строки ввода**

Если необходимо ввести из формы в программу или вывести в форму информацию, которая вмещается в одну строку, используют окно однострочного редактора текста, представляемого элементом управления TextBox.

Создадим поле для ввода имени пользователя. Выберем на панели инструментов пиктограмму с названием «TextBox», щелкните мышью в том месте формы, где хотите ее разметить. После размещения элемента можно в тексте программы использовать переменную `textBox1`, которая соответствует добавленному элементу управления. В этой переменной, в свойстве `Text` будет содержаться строка символов (типа `string`) и отображаться в соответствующем окне TextBox. С помощью окна свойств можно установить шрифт и размер символов, отражаемых в строке TextBox (свойство `Font`).

## Размещение надписей

На форме могут размещаться пояснительные надписи. Для размещения таких надписей на форму используется элемент Label. Выберите на панели инструментов пиктограмму с названием Label, щелкните на ней мышью. После этого в нужном месте формы щелкните мышью, появится надпись label1. Щелкнув на ней мышью, можно отрегулировать размер и, изменив свойство Text в окне свойств, введите строку, например, «Введите свое имя:», а также выберите размер символов (свойство Font). Добавленный элемент представлен в форме на рисунке 3.5.

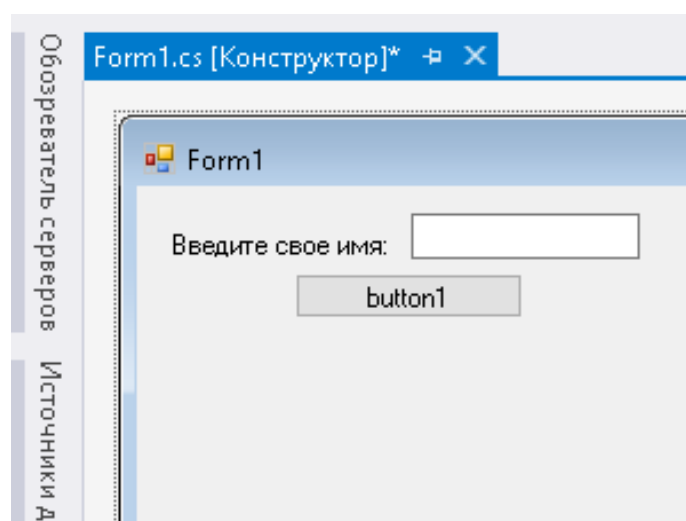


Рисунок 3.2 – Форма с добавленным элементом Label

В тексте программы можно обращаться к новой переменной типа Label. В ней хранится пояснительная строка, которую можно изменять в процессе работы программы.

## Написание программы обработки события

С каждым элементом управления на форме и с самой формой могут происходить события во время работы программы. Например, с кнопкой может произойти событие – нажатие кнопки, а с окном, которое проектируется с помощью формы, может произойти ряд событий: создание окна, изменение размера окна, щелчок мыши на окне и т.п. Эти события могут обрабатываться программно. Для обработки таких событий используются специальные методы – обработчики событий. Создать такие методы можно двумя способами.

Первый способ – создать обработчик для события по умолчанию. Например, при нажатии кнопки на форме таким образом создается обработчик события нажатия.

### Написание программы обработки события нажатия кнопки

Поместите на форму кнопку, которая описывается элементом управления Button. С помощью окна свойств измените заголовок (Text) на слово «Привет» или другое по вашему желанию. Отрегулируйте положение и размер кнопки.

После этого два раза щелкните мышью на кнопке, появится текст программы:

```
private void button1_Click(object sender, EventArgs e)
{
}
```

Этот метод и будет являться обработчиком события нажатия кнопки. Вы можете добавлять свой код между скобками {...}. Например, наберите:

```
private void button1_Click(object sender, EventArgs e)
{
    MessageBox.Show("Привет " + textBox1.Text + "!");
}
```

При нажатии на кнопку выведется окно с сообщением «Привет», которое соединено с текстом, записанным пользователем в textBox1 (Рисунок 3.6).

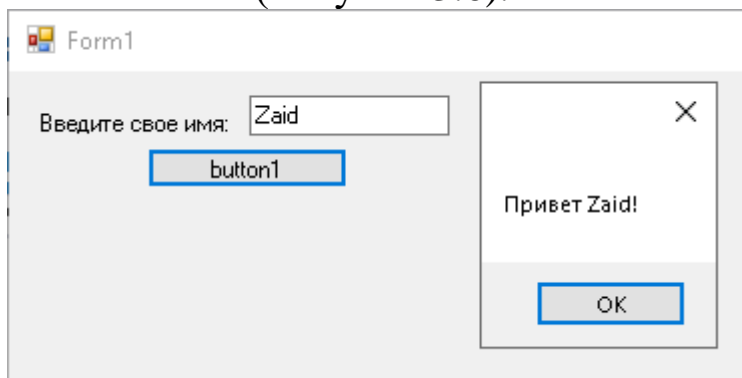



Рисунок 3.3 – Окно с сообщением, появившееся при нажатии кнопки button1

### Написание программы обработки события загрузки формы

Второй способ создания обработчика события заключается в выборе соответствующего события для выделенного элемента на форме. При этом используется окно свойств и его закладка . Рассмотрим этот способ. Выделите форму щелчком по ней,

чтобы вокруг нее появилась рамка из точек. В окне свойств найдите событие Load. Щелкните по данной строчке дважды мышкой. Появится метод:

```
private void Form1_Load(object sender, EventArgs e)
{
}
```

Между скобками {...} вставим текст программы:

```
private void Form1_Load(object sender, EventArgs e)
{
    BackColor = Color.AntiqueWhite;
}
```

Свойство BackColor позволяет изменить цвет фона. Результат выполнения данного метода представлен на рисунке 3.7.

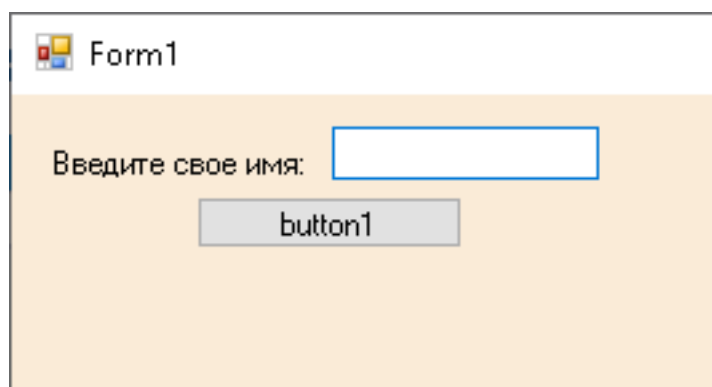


Рисунок 3.4 – Форма с измененным фоном

Каждый элемент управления имеет свой набор обработчиков событий, однако некоторые из них присущи большинству элементов управления. Наиболее часто применяемые события описаны ниже:

1) Load – событие, возникающее при загрузке формы. В обработчике данного события следует задавать действия, которые должны происходить в момент создания формы, например, установка начальных значений.

2) KeyPress – событие, возникающее при нажатии кнопки на клавиатуре. Параметр e.KeyChar имеет тип char и содержит код нажатой клавиши (например, клавиша Enter клавиатуры имеет код #13, клавиша Esc - #27 и т.д.). Обычно это событие используется в том случае, когда необходима реакция на нажатие одной из клавиш.

3) **KeyDown** – событие, возникающее при нажатии кнопки на клавиатуре. Обработчик этого события получает информацию о нажатой клавише и состоянии клавиш Shift, Alt и Ctrl, а также о нажатой кнопке мыши. Информация о клавише передается параметром `e.KeyCode`, который представляет собой перечисление `Keys` с кодами всех клавиш, а информацию о клавишах-модификаторах Shift и др. можно узнать из параметра `e.Modifiers`.

4) **KeyUp** – является парным событием для **KeyDown** и возникает при отпускании ранее нажатой клавиши.

5) **Click** – возникает при нажатии кнопкой мыши в области элемента управления.

6) **DoubleClick** – возникает при двойном нажатии кнопки мыши в области элемента управления.

**Важное примечание** - если какой-то обработчик был добавлен по ошибке или больше не нужен, то для его удаления нельзя просто удалить программный код обработчика. Сначала нужно удалить строку с именем обработчика в окне свойств в закладке с событиями. В противном случае программа может перестать компилироваться и даже отображать форму в дизайнера VS.

### **Динамическое изменение свойств**

Свойства элементов на окне могут быть изменены динамически во время выполнения программы. Например, можно изменять текст надписи или цвет формы. Изменение свойств происходит внутри обработчика события (например, обработчика события нажатия на кнопку). Для этого используют оператор присвоения вида:

`<имя элемента>.<свойство> = <значение>;`

Например,

`label1.Text = "Hello";`

`<имя элемента>` определяется на этапе проектирования формы, при размещении элемента управления на форме. Например, при размещении на форме ряда элементов `TextBox`, эти элементы получают имена `textBox1`, `textBox2`, `textBox3` и т.д., которые могут быть заменены в окне свойств в свойстве (`Name`) для текущего элемента. Список свойств для конкретного элемента можно посмотреть в окне свойств.



Если требуется изменить свойства формы, то никакое имя элемента перед точкой вставлять не нужно, как и саму точку. Например, чтобы задать цвет формы, нужно просто написать:  
`BackColor = Color.Green;`

### **Ввод и вывод данных в программу**

Рассмотрим один из способов ввода данных через элементы, размещенные на форме. Для ввода данных чаще всего используют элемент управления `TextBox`, через обращение к его свойству `Text`. Свойство `Text` хранит в себе строку введенных символов. Поэтому данные можно считать следующим образом:

```
private void button1_Click(object sender, EventArgs e)
{
    string s = textBox1.Text;
}
```

Однако со строкой символов трудно производить арифметические операции, поэтому лучше всего при вводе числовых данных перевести строку в целое или вещественное число. Для этого у типов `int` и `double` существуют методы `Parse` для преобразования строк в числа. С этими числами можно производить различные арифметические действия. Таким образом, предыдущий пример можно переделать следующим образом:

Перед выводом числовые данные следует преобразовать назад в строку. Для этого у каждой переменной существует метод `ToString()`, который возвращает в результате строку с символьным представлением значения. Вывод данных можно осуществлять в элементы `TextBox` или `Label`, используя свойство `Text`. Например,

```
private void button1_Click(object sender, EventArgs e)
{
    string s = textBox1.Text;
    int a = int.Parse(s);
    int b = a * a;
    label1.Text = b.ToString();
}
```

### **Кнопки-переключатели**

При создании программ в Visual Studio для организации разветвлений часто используются элементы управления в виде кнопок-переключателей. Кнопки-переключатели представлены в

панели инструментов как `RadioButton`. Состояние такой кнопки (включено-выключено) визуально отражается на форме, а в программе можно узнать его помощью свойства `Checked`: если кнопка включена, это свойство будет содержать `True`, в противном случае `False`. Если пользователь выбирает один из вариантов переключателя в группе, все остальные автоматически отключаются.

Группируются радиокнопки с помощью какого-либо контейнера – часто это бывает элемент `GroupBox`. Радиокнопки, размещенные в различных контейнерах, образуют независимые группы.

Контейнер с кнопками переключателями представлен на рисунке 3.8.

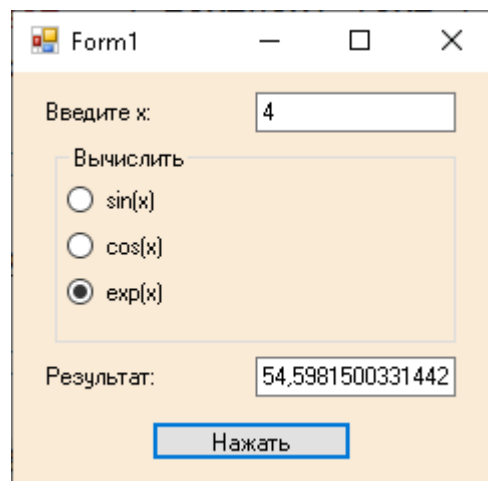


Рисунок 3.5. Группа кнопок-переключателей

Взаимодействие с кнопками-переключателями описывает следующий код:

```
if(radioButton1.Checked)
{
    textBox1.Text = Convert.ToString(Math.Sin(Convert.ToDouble(textBox2.Text)));
}
else if (radioButton2.Checked)
{
    textBox1.Text = Convert.ToString(Math.Cos(Convert.ToDouble(textBox2.Text)));
}
else if (radioButton3.Checked)
{
    textBox1.Text = Convert.ToString(Math.Exp(Convert.ToDouble(textBox2.Text)));
}
```

### Флажок для множественного выбора

Элемент CheckBox или флажок предназначен для установки одного из двух значений: отмечен или не отмечен. Чтобы отметить флажок, надо установить у его свойства Checked значение true.

Кроме свойства Checked у элемента CheckBox имеется свойство CheckState, которое позволяет задать для флажка одно из трех состояний - Checked (отмечен), Indeterminate (флажок не определен - отмечен, но находится в неактивном состоянии) и Unchecked (не отмечен).

Также следует отметить свойство AutoCheck - если оно имеет значение false, то мы не можем изменять состояние флажка. По умолчанию оно имеет значение true. Форма с элементами-флажками представлена на рисунке 3.9.

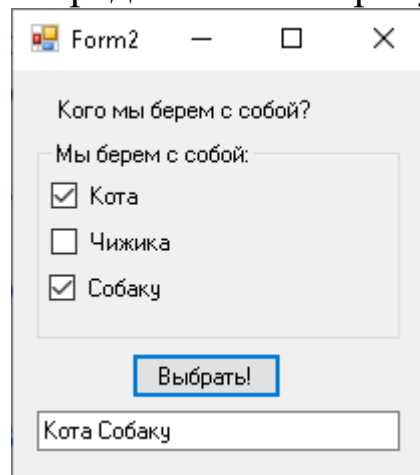


Рисунок 3.6. Форма с CheckBox

Взаимодействие с CheckBox позволяет описать следующий

код:

```
private void button1_Click(object sender, EventArgs e) //кнопка "Выбрать"
{
    textBox1.Text = "";
    switch(checkBox1.Checked)
    {
        case true:
        {
            textBox1.Text += checkBox1.Text + " ";
            break;
        }
        case false:
        {
            break;
        }
    }
}
```

```

    }
    switch(checkBox2.Checked)
    {
        case true:
        {
            textBox1.Text += checkBox2.Text + " ";
            break;
        }
        case false:
        {
            break;
        }
    }
    switch(checkBox3.Checked)
    {
        case true:
        {
            textBox1.Text += checkBox3.Text + " ";
            break;
        }
        case false:
        {
            break;
        }
    }
}

```

### Элемент **ListBox**

Элемент **ListBox** представляет собой простой список. Ключевым свойством этого элемента является свойство **Items**, которое как раз и хранит набор всех элементов списка.

Элементы в список могут добавляться как во время разработки, так и программным способом. В **Visual Studio** в окне **Properties** (Свойства) для элемента **ListBox** мы можем найти свойство **Items**. После двойного щелчка на свойство нам отобразится окно для добавления элементов в список (Рисунок 10).

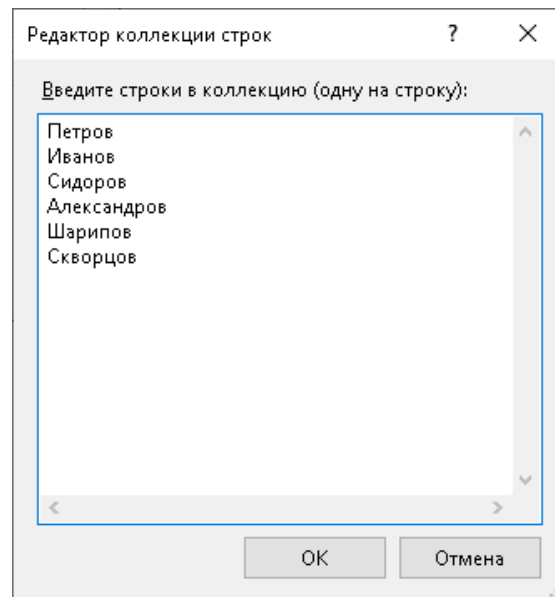


Рисунок 3.7 – Редактор коллекции строк в ListBox

В пустое поле мы вводим по одному элементу списка - по одному на каждой строке. После этого все добавленные нами элементы окажутся в списке, и мы сможем ими управлять. Создадим форму (Рисунок 11) для управления элементами формы. В качестве функционала добавим возможность добавления элемента в конец списка, удаление и отображение выделенного элемента.

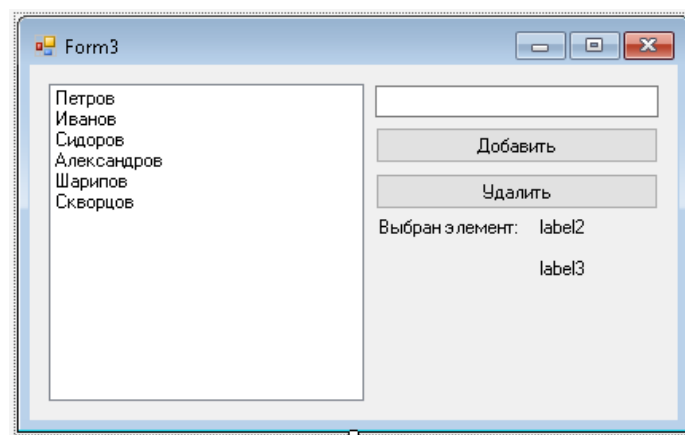


Рисунок 3.8 – Макет формы для управления содержимым списка

Итак, все элементы списка входят в свойство `Items`, которое представляет собой коллекцию. Для добавления нового элемента в эту коллекцию, а значит и в список, надо использовать метод `Add`, например: `listBox1.Items.Add("Новый элемент");`. При использовании этого метода каждый добавляемый элемент добавляется в конец списка.

Можно добавить сразу несколько элементов, например, массив. Для этого используется метод `AddRange`:

```
string[] students = {"Кондратьев", "Дормидонтова", "Валеева"};
listBox1.Items.AddRange(students);
```

В отличие от простого добавления вставка производится по определенному индексу списка с помощью метода `Insert`:

```
listBox1.Items.Insert(listBox1.Items.Count, textBox1.Text);
```

Свойство `listBox1.Items.Count` определяет текущее количество элементов в списке. В вышеописанной строчке кода вставка элемента будет производиться постоянно в конец списка.

Для удаления элемента по его тексту используется метод `Remove`:

```
listBox1.Items.Remove("Петров");
```

Чтобы удалить элемент по его индексу в списке, используется метод `RemoveAt`:

```
listBox1.Items.RemoveAt(1);
```

В качестве параметра передается индекс удаляемого элемента.

Кроме того, можно очистить сразу весь список, применив метод `Clear`:

```
listBox1.Items.Clear();
```

Используя индекс элемента, можно получить доступ к самому элементу в списке. Например, получим первый элемент списка:

```
string firstElement = listBox1.Items[0];
```

При выделении элементов списка мы можем ими управлять как через индекс, так и через сам выделенный элемент. Получить выделенные элементы можно с помощью следующих свойств элемента `ListBox`:

- `SelectedIndex`: возвращает или устанавливает номер выделенного элемента списка. Если выделенные элементы отсутствуют, тогда свойство имеет значение -1
- `SelectedIndices`: возвращает или устанавливает коллекцию выделенных элементов в виде набора их индексов
- `SelectedItem`: возвращает или устанавливает текст выделенного элемента
- `SelectedItems`: возвращает или устанавливает выделенные элементы в виде коллекции

По умолчанию список поддерживает выделение одного элемента. Чтобы добавить возможность выделения нескольких

элементов, надо установить у его свойства `SelectionMode` значение `MultiSimple`.

Чтобы выделить элемент программно, надо применить метод `SetSelected(int index, bool value)`, где `index` - номер выделенного элемента. Если второй параметр - `value` имеет значение `true`, то элемент по указанному индексу выделяется, если `false`, то выделение наоборот скрывается:

```
listBox1.SetSelected(2, true); // будет выделен третий элемент
```

Чтобы снять выделение со всех выделенных элементов, используется метод `ClearSelected`.

Из всех событий элемента `ListBox` надо отметить в первую очередь событие `SelectedIndexChanged`, которое возникает при изменении выделенного элемента. Рассмотрим реализацию обработчика события на примере нашего задания.

```
private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
{
    label2.Text = Convert.ToString(listBox1.SelectedItem);
    label3.Text = Convert.ToString(listBox1.SelectedIndex);
}
```

Обработчик события изменяет текст в надписях. В `label2` выводится текст выбранного пользователем элемента, а в `label3` – его индекс.

При нажатии на кнопку «Добавить» будет происходить добавление нового элемента в конец списка. Конец элемента будет определять индекс, равный `listBox1.Items.Count` – текущему количеству элементов в списке. Текстовое наполнение нового элемента будет извлекаться из данных, введенных в `textBox1`.

```
private void button1_Click(object sender, EventArgs e)
{
    listBox1.Items.Insert(listBox1.Items.Count, textBox1.Text);
}
```

Кнопка «Удалить» будет производить удаление выбранного в списке элемента. `listBox1.SelectedIndex` будет возвращать индекс выбранного пользователем элемента.

```
private void button2_Click(object sender, EventArgs e)
{
    listBox1.Items.RemoveAt(listBox1.SelectedIndex);
}
```

### **ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №3.**

1. Реализуйте игру «Найди пару» с помощью Windows-Forms C#.
2. Добавьте дополнительные возможности, чтобы сделать эту игру более сложной и интересной.
3. Замените значки и цвета на другие.
4. Добавьте таймер игры, который отслеживает время, необходимое игроку для победы. Вы можете добавить метку, чтобы отобразить время, затраченное на форму. Разместите ее над `TableLayoutPanel`.
5. Добавьте в форму еще один таймер, чтобы иметь возможность отслеживать время. Следующий код служит для запуска таймера, когда игрок начинает игру, и остановки таймера после сопоставления последних двух значков.
6. Добавьте звуки, которые будут воспроизводиться при нахождении игроком пары, отображении двух несовпадающих значков и повторном сокрытии значков программой. Для воспроизведения звуков можно использовать пространство имен `System.Media`.
7. Сделайте игру труднее, увеличив игровое поле. Необходимо сделать больше, чем просто добавить строки и столбцы в `TableLayoutPanel`. Вы должны также учитывать количество создаваемых значков.
8. Сделайте игру интереснее, скрывая первый значок, если игрок медлителен.



## ПРАКТИЧЕСКАЯ РАБОТА №4. СВОЙСТВА И МЕТОДЫ ОБЪЕКТОВ. РЕАКЦИЯ ОБЪЕКТОВ НА СОБЫТИЯ

**Цель работы:** Изучение основных событий. Знакомство с написанием процедур отклика на события. Ссылки на свойства. Методы.

### ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Для каждого объекта существует ряд возможных событий. Программный код отклика на эти события содержится в процедурах. Основные события приведены в таблице 4.1.

Таблица 4.1 – События

Событие	Описание
Click (щелчок)	Происходит при щелчке пользователя кнопкой мыши по объекту
Double Click (двойной щелчок)	Происходит при двойном щелчке пользователя кнопкой мыши по объекту
Mouse Down	Происходит при нажатии кнопки мыши
Mouse Up	Происходит при отпускании кнопки мыши
Mouse Move	Происходит при прохождении указателя мыши по объекту
Drag Over	Происходит при перетаскивании элемента формы через объект
Drag Drop	Происходит в конце операции перетаскивания, когда кнопка мыши отпускается
Got Focus	Происходит, когда элемент управления получает активность
Lost Focus	Происходит, когда элемент управления теряет активность
Load	Происходит при каждой загрузке формы в память
Initialize	Возникает в момент создания экземпляра формы (до ее загрузки в память компьютера и отображения на экране)

Процедуры создаются в кодовом окне. Каждая процедура должна иметь имя, которое должно состоять из имени объекта, символа подчеркивания и имени события. Процедура записывается в следующем виде:

```
Private Sub объект_событие()
    операторы
    .....
End Sub
```

В процедуре можно использовать любые свойства элементов управления. Для того чтобы в процессе работы программы изменить свойство объекта, необходимо записать выражение следующего вида:  
объект . свойство = значение

Пример: При щелчке мыши по командной кнопке «ОК» (cmdOK) ярлык (lblText) изменяет свой текст на «Добро пожаловать в VB». Private Sub cmdOK\_Click()  
lblText.Caption=«Добро пожаловать в VB»  
End Sub

Каждый объект имеет некоторое количество доступных ему методов. Методы – это встроенные процедуры, оказывающие некоторое действие на объект. Некоторые методы приведены в таблице 4.2.

Таблица 4.2 – Методы

Метод	Описание
Set Focus	Активизация объекта
Show	Загрузка в память формы и вывод ее на экран
Hide	Соккрытие формы с экрана
AddItem элемент	Применяется для окон списков. Добавление элемента в список во время работы программы
RemoveItem элемент	Применяется для окон списков. Удаляет элемент из списка

Для того чтобы выполнить один из методов необходимо записать выражение следующего вида: объект.метод

Пример: Добавление фамилий студентов в комбинированный список. Выбор фамилии из списка (рис. 4.1).

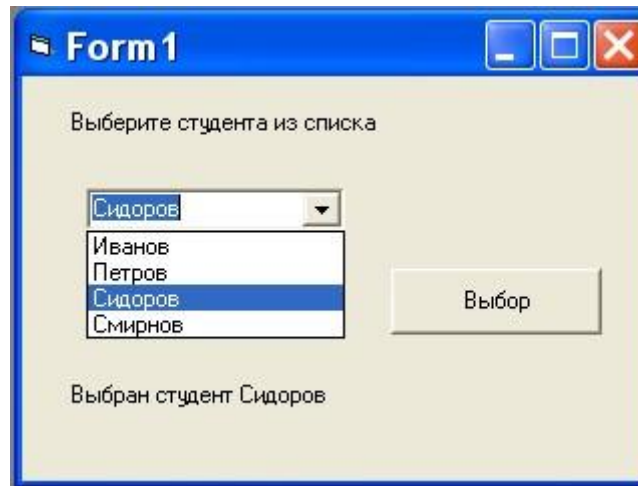


Рисунок 4.1 – Форма

```
Private Sub form_initialize ()
cmbStudent.AddItem «Иванов»
cmbStudent.AddItem «Петров»
cmbStudent.AddItem «Сидоров»
cmbStudent.AddItem «Смирнов»
End Sub
Private Sub vibor_Click()
Label1.Caption="Выбран студент" & cmbStudent.Text End Sub
```

### ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №4

1. К формам, созданным в предыдущей практической работе, добавьте кнопки перехода от одной формы к другой. Напишите программный код, позволяющий проверить правильность ввода пароля. Добавьте возможность выхода из приложения при нажатии на кнопку «Выход», размещенную на последней форме.

2. Разработайте интерфейс формы, напишите программный код, позволяющий все данные, вводимые пользователем в текстовое окно, автоматически при нажатии на кнопку «ОК» отображать на ярлыке.

3. Создайте блок кнопок - «Красный», «Желтый», «Зеленый», «Синий», «Белый», «Черный». При нажатии на кнопку форма окрашивается соответствующим цветом.

4. При нажатии на кнопку «Min» размер формы уменьшается на 10 твилов, «Max» – увеличивается на 10 твилов (1 пункт = 20 твилов).

Пункт – единица, используемая для измерения размера шрифта).

5. Создать следующую форму (рис. 4.2) для ввода текста:

Рисунок 4.2 – Форма

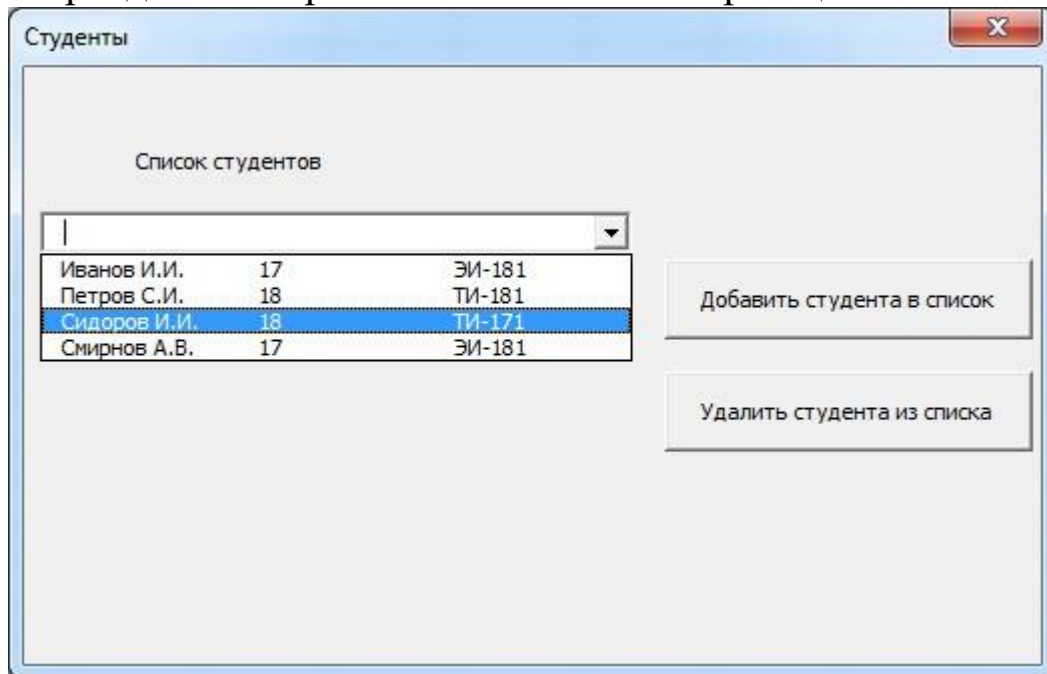
6. Создать форму для сбора данных (рис. 4.3). После нажатия на кнопку “Далее”, введенные данные выводятся на экран с помощью функции MsgBox().

Рисунок 4.3 – Форма

7. Создайте форму для добавления/удаления данных о студентах в комбинированный список, состоящий из 3 колонок (рис. 4.4).

Примечание: для добавления нового элемента используется метод `AddItem`, для записи данных в колонки необходимо использовать свойство `List`. Например, для помещения числа «17» в 0-й строку и 1-ю колонку комбинированного списка `cmbStudent` необходимо записать: `cmbStudent.List(0, 1) = "17"`.

8. Создать проект “Выбор шрифта”, позволяющий выбрать с помощью списков тип шрифта, размер, начертание (свойства): жирный, подчёркнутый, курсив. Выбранные параметры должны применяться к метке Образец.



Студенты

Список студентов

Имя	Возраст	Идентификатор
Иванов И.И.	17	ЭИ-181
Петров С.И.	18	ТИ-181
Сидоров И.И.	18	ТИ-171
Смирнов А.В.	17	ЭИ-181

Добавить студента в список

Удалить студента из списка

Рисунок 4.4 – Форма

## ПРАКТИЧЕСКАЯ РАБОТА №5. ПОСТРОЕНИЕ ВЫРАЖЕНИЙ

**Цель работы:** ознакомиться с правилами записи операций и выражений в языке программирования C#, получить основные навыки работы с ней, освоить приёмы создания, компиляции и исполнения программы.

### ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

#### 1. Инкремент (++) и декремент (--).

Эти операции имеют две формы записи - префиксную, когда операция записывается **перед** операндом, и постфиксную - операция записывается **после** операнда.

Префиксная операция инкремента/декремента увеличивает/уменьшает свой операнд и возвращает измененное значение как результат.

Постфиксные версии инкремента и декремента возвращают первоначальное значение операнда, а затем изменяют его.

Рассмотрим пример использования операции инкремента и декремента (рис. 5.1)

Ссылка: 0

```
static void Main(string[] args)
{
    int i = 3, j = 4;
    Console.WriteLine("{0} {1}", i, j);
    Console.WriteLine("{0} {1}", ++i, --j);
    Console.WriteLine("{0} {1}", i++, j--);
    Console.WriteLine("{0} {1}", i, j);
    Console.ReadKey();
}
```

Рисунок 5.1 – Код программы

Результат работы данной программы показан на рис. 5.2.

```
3 4
4 3
4 3
5 2
```

Рисунок 5.2 – Результат работы программы

**Замечание.** Префиксная версия требует существенно меньше действий: она изменяет значение переменной и запоминает результат в ту же переменную. Постфиксная операция должна отдельно сохранить исходное значение, чтобы затем вернуть его как результат. Для сложных типов подобные дополнительные действия могут оказаться трудоемки. Поэтому постфиксную форму имеет смысл использовать только при необходимости.

## 2. Отрицание:

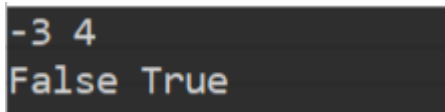
- Арифметическое отрицание (-) - меняет знак операнда на противоположный.
- Логическое отрицание (!) - определяет операцию инверсия для логического типа.

Рассмотрим пример использования данных операций (рис. 5.3 и 5.4)

Ссылка: 0

```
static void Main(string[] args)
{
    int i = 3, j = -4;
    bool a = true, b = false;
    Console.WriteLine("{0} {1}", -i, -j);
    Console.WriteLine("{0} {1}", !a, !b);
    Console.ReadKey();
}
```

Рисунок 5.3 – Код программы



```
-3 4
False True
```

Рисунок 5.4 – Результат работы программы

## 3. Умножение (\*), деление (/) и остаток от деления (%).

Операции умножения и деления применимы для целочисленных и вещественных типов данных. Для других типов эти операции применимы, если для них возможно неявное преобразование к целым или вещественным типам. При этом тип результата равен «наибольшему» из типов операндов, но не менее int. Если оба операнда при делении целочисленные, то и результат тоже целочисленный.

Рассмотрим пример использования данных операций (рис. 5.5 и 5.6).

```

Ссылка: 0
static void Main(string[] args)
{
    int i = 100, j = 15;
    double a = 14.2, b = 3.5;
    Console.WriteLine("{0} {1} {2}", i * j, i / j, i % j);
    Console.WriteLine("{0} {1} {2}", a * b, a / b, a % b);
    Console.ReadKey();
}

```

Рисунок 5.5 – Код программы

```

1500 6 10
49,7 4,05714285714286 0,199999999999999

```

Рисунок 5.6 – Результат работы программы

#### 4. Сложение (+) и вычитание (-).

Операции сложения и вычитания применимы для целочисленных и вещественных типов данных. Для других типов эти операции применимы, если для них возможно неявное преобразование к целым или вещественным типам.

#### 5. Операции отношения (<, <=, >, >=, ==, !=).

Операции отношения сравнивают значения левого и правого операндов. Результат операции логического типа:

- true - если значения совпадают.
- false - в противном случае.

Рассмотрим пример использования данных операций (рис. 5.7 и 5.8).

```

Ссылка: 0
static void Main(string[] args)
{
    int i = 15, j = 15;
    Console.WriteLine(i < j); //меньше
    Console.WriteLine(i <= j); //меньше или равно
    Console.WriteLine(i > j); //больше
    Console.WriteLine(i >= j); //больше или равно
    Console.WriteLine(i == j); //равно
    Console.WriteLine(i != j); //не равно
    Console.ReadKey();
}

```

Рисунок 5.7 – Код программы

```

False
True
False
True
True
False

```

Рисунок 5.8 – Результат работы программы

#### 6. Логические операции && и ||.

Применяются к операндам логического типа.



– Результат логической операции **И** (**&&**) имеет значение истина тогда и только тогда, когда оба операнда принимают значение истина.

– Результат логической операции **ИЛИ** (**||**) имеет значение истина тогда и только тогда, когда хотя бы один из операндов принимает значение истина.

Рассмотрим пример использования данных операций (рис. 5.9 и 5.10).

```

Ссылка: 0
static void Main(string[] args)
{
    Console.WriteLine("x\ty\tx и y\tx или y");
    Console.WriteLine("{0}\t{1}\t{2}\t{3}", false, false, false && false, false || false);
    Console.WriteLine("{0}\t{1}\t{2}\t{3}", false, true, false && true, false || true);
    Console.WriteLine("{0}\t{1}\t{2}\t{3}", true, false, true && false, true || false);
    Console.WriteLine("{0}\t{1}\t{2}\t{3}", true, true, true && true, true || true);
    Console.ReadKey();
}

```

Рисунок 5.9 – Код программы

x	y	x и y	x или y
False	False	False	False
False	True	False	True
True	False	False	True
True	True	True	True

Рисунок 5.10 – Результат работы программы

## 7. Тернарный оператор.

Формат: (<операнд1>)? <операнд2> : <операнд3> ;

Операнд1 – это логическое выражение, которое оценивается с точки зрения его эквивалентности константам true и false.

Если результат вычисления операнда1 равен true, то результатом условной операции будет значение операнда2, иначе - операнда3.

Фактически условная операция является сокращенной формой условного оператора if, который будет рассмотрен позже.

```
static void Main(string[] args)
{
    int x = 5; int y = 10;
    int max = (x > y) ? x : y;
    Console.WriteLine(max);
    Console.ReadKey();
}
```

Рисунок 5.11 – Код программы

## 8. Операции присваивания: =, +=, -= и т.д.

Формат операции простого присваивания ( = ):

операнд\_2 = операнд\_1;

В результате выполнения этой операции вычисляется значение операнда\_1, и результат записывается в операнд\_2.

Возможно связать воедино сразу несколько операторов присваивания, записывая такие цепочки: a=b=c=100. Выражение такого вида выполняется справа налево: результатом выполнения c=100 является число 100, которое затем присваивается переменной b, результатом чего опять является 100, которое присваивается переменной a.

Кроме простой операции присваивания существуют сложные операции присваивания, например, умножение с присваиванием ( \*= ), деление с присваиванием ( /= ), остаток от деления с присваиванием ( %= ), сложение с присваиванием ( += ), вычитание с присваиванием ( -= ) и т.д.

В сложных операциях присваивания, например, при сложении с присваиванием, к операнду\_2 прибавляется операнд\_1, и результат записывается в операнд\_2.

То есть, выражение c += a это более компактная запись выражения c = c + a.

Кроме того, сложные операции присваивания позволяют сгенерировать более эффективный код, за счет того, что в простой операции присваивания для хранения значения правого операнда создается временная переменная, а в сложных операциях присваивания значение правого операнда сразу записывается в левый операнд.

Рассмотренные операции приведены с учетом убывания приоритета. Если в одном выражении соседствуют операции одного приоритета, то операции присваивания и условная

операции выполняются справа налево, а остальные, наоборот. Если необходимо изменить порядок выполнения операций, то в выражении необходимо поставить круглые скобки.

## 9. Выражения и преобразование типов

Выражение – это синтаксическая единица языка, определяющая способ вычисления некоторого значения. Выражения состоят из операндов, операций и скобок. Каждый операнд является выражением или одним из его частных случаев - константой, переменной или функций.

$(a + 0.12)/6$

$x \&\& y \parallel !z$

$(t * \text{Math.Sin}(x) - 1.05e4) / ((2 * k + 2) * (2 * k + 3))$

Операции выполняются в соответствии с приоритетами. Для изменения порядка выполнения операций используются круглые скобки. Если в одном выражении записано несколько операций одинакового приоритета, то унарные операции, условная операция и операции присваивания выполняются справа налево, остальные - слева направо.

Например,

$a = b = c$  означает  $a = (b = c)$ ,

$a + b + c$  означает  $(a + b) + c$ .

В выражение могут входить операнды различных типов.

Если операнды имеют одинаковый тип, то результат операции будет иметь тот же тип.

Если операнды разного типа, то перед вычислениями выполняются преобразования более коротких типов в более длинные для сохранения значимости и точности.

## ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №5

### Вариант 1.

**Задание.** Написать программу, которая реализует диалог с пользователем:

1. Запрашивает с клавиатуры два целых числа, и выводит на экран сумму данных чисел (рис. 5.12):

```
a= 23
b= 43
23+43=66
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.12 – Результат работы программы

2. Запрашивает с клавиатуры три целых числа, и выводит на экран сумму данных чисел в прямом и обратном порядке (рис. 5.13):

```
a= 12
b= 34
12+34=34+12
Для продолжения нажмите любую клавишу . . . _
```

Рисунок 5.13 – Результат работы программы

3. Запрашивает с клавиатуры три целых числа, и выводит на экран сумму данных чисел (рис. 5.14):

```
a= 24
b= 5
c= 36
24+5+36=65
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.14 – Результат работы программы

4. Запрашивает с клавиатуры два вещественных числа, и выводит на экран произведение данных чисел (вещественные числа выводятся с точностью до 1 знака после запятой) (рис. 5.15):

```
a= 3,45
b= 12,1
3,5*12,1=41,7
Для продолжения нажмите любую клавишу . . . _
```

Рисунок 5.15 – Результат работы программы

5. Запрашивает с клавиатуры два вещественных числа, и выводит на экран результат деления первого числа на второе (вещественные числа выводятся с точностью до 3 знаков после запятой) (рис. 5.16):

```
a= 345,6
b= 12,345
345,600/12,345=27,995
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.16 – Результат работы программы

## Вариант 2.

**Задание.** Написать программу, которая реализует диалог с пользователем:

1. Запрашивает с клавиатуры три вещественных числа, и выводит на следующее сообщение (вещественные числа выводятся с точностью до 2 знаков после запятой) (рис. 5.17):

```

a= 12,4
b= 2,567
c= 100
<12,40+2,57>+100,00=12,40+<2,57+100,00>
Для продолжения нажмите любую клавишу . . .

```

Рисунок 5.17 – Результат работы программы

2. Запрашивает с клавиатуры два целых числа, и выводит на экран произведение данных чисел (рис. 5.18):

```

a= 23
b= 43
23*43= 989
Для продолжения нажмите любую клавишу . . .

```

Рисунок 5.18 – Результат работы программы

3. Запрашивает с клавиатуры три целых числа, и выводит на экран произведение данных чисел в прямом и обратном порядке (рис. 5.19):

```

a= 12
b= 34
12*34=34*12
Для продолжения нажмите любую клавишу . . . _

```

Рисунок 5.19 – Результат работы программы

4. Запрашивает с клавиатуры два вещественных числа, и выводит на экран сумму данных чисел (вещественные числа выводятся с точностью до 2 знаков после запятой) (рис. 5.20):

```

a= 3,45
b= 12,1
3,5+12,1=15,55
Для продолжения нажмите любую клавишу . . . _

```

Рисунок 5.20 – Результат работы программы

5. Запрашивает с клавиатуры два вещественных числа, и выводит на экран результат деления второго числа на первое (вещественные числа выводятся с точностью до 2 знаков после запятой) (рис. 5.21):

```

a= 12,345
b= 345,6
345,60/12,345=27,99
Для продолжения нажмите любую клавишу . . .

```

Рисунок 5.21 – Результат работы программы

### Вариант 3.

**Задание.** Написать программу, которая реализует диалог с пользователем:

1. Запрашивает с клавиатуры три вещественных числа и выводит следующее сообщение  $(a+(b+c))=(a+c+b)$  (вещественные числа выводятся с точностью до 4 знаков после запятой) (рис. 5.22):

```
a= 12,4
b= 2,567
c= 100
<12,4000+<2,5700+100,0000>=<12,4000+100,0000+2,5700>
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.22 – Результат работы программы

2. Запрашивает с клавиатуры четыре вещественных числа, и выводит на экран результат деления первого числа на второе плюс третьего на четвертое (вещественные числа выводятся с точностью до 2 знаков после запятой) (рис. 5.23):

```
a= 345,6
b= 12,34
c= 13,02
d= 3,1
345,60/12,34 + 13,02/3,10=28,97
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.23 – Результат работы программы

3. Запрашивает с клавиатуры два целых числа, и выводит на экран результат их суммы, разности и произведения (рис. 5.24):

```
a=15
b=84
a+b=15+84=99      a-b=15-84=-69      a*b=15*84=1260
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.24 – Результат работы программы

4. Переменной вещественного типа x присваивает значение, равное полусумме значений a и b с точностью 3 знака после запятой (рис. 5.25):

```
a=3,14
b=51,2
3,14/2+51,2/2=27,170
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.25 – Результат работы программы



5. Утраивает значение вещественной переменной  $x$  и выполняет деление на целое число  $y$ , результат округлить до 4 знаков после запятой (рис. 5.26):

```
x=8,265539
y=2
округленный с точностью до 4 знаков результат: 24,796617/2=12,3983
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.26 – Результат работы программы

#### Вариант 4.

**Задание.** Написать программу, которая реализует диалог с пользователем:

1. Переменной вещественного типа  $x$  присвоить значение на  $\pi$  меньше исходного, результат округлить до 8 знаков после запятой (рис. 5.27):

```
x=6,9830915838
PI=3,14159265358979
x-PI=6,9830915838-3,14159265358979=3,84149893
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.27 – Результат работы программы

2. Меняет знак  $y$  переменной  $x$  на противоположный и выполняет умножение на вещественное число  $y$  (рис. 5.28):

```
x=2,25
y=6
-x=-2,25      -x*y=-2,25*6=-13,5
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.28 – Результат работы программы

3. Запрашивает с клавиатуры три целых числа, и выводит на экран результат модуля разности первого, второго и третьего чисел (рис. 5.29):

```
x=-35
y=4
z=10
k=-49
!x-y-z!=|-35-4-10|=49
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.29 – Результат работы программы

4. Переменной целого типа  $x$  присвоить значение, равное половине произведения значений  $a$ ,  $b$ ,  $c$  (рис. 5.30):

```

a=34
b=-2
c=61
x=(a*b*c)/2=(34*-2*61)=-2074
Для продолжения нажмите любую клавишу . . .

```

Рисунок 5.30 – Результат работы программы

5. Запрашивает с клавиатуры три целых числа, и выводит на экран результат деления первого числа на  $\pi$  (с точностью 4 знака после запятой), сумму данных чисел и результат деления третьего числа на  $E$  (с точностью 3 знака после запятой) (рис. 5.31):

```

a=34
b=65
c=24
a/PI=34/3,1416=10,8225   a+b+c=34+65+24=123   c/E=24/2,718=8,829
Для продолжения нажмите любую клавишу . . .

```

Рисунок 5.31 – Результат работы программы

### Вариант 5.

**Задание.** Написать программу, которая реализует диалог с пользователем:

1. Запрашивает с клавиатуры четыре целых числа, и выводит на экран результат умножения первого числа на третье минус произведение второго числа на четвертое (рис. 5.32):

```

a=9
b=5
c=11
d=3
a*c-b*d=9*11+5*3=84
Для продолжения нажмите любую клавишу . . .

```

Рисунок 5.32 – Результат работы программы

2. Запрашивает с клавиатуры два вещественных числа, и выводит на экран результат их умножения друг на друга в два столбика: первый столбик – вывод результата с точностью 3 знака после запятой, второй столбик – с точностью 5 знаков после запятой (рис. 5.33):

```

a=5,375864
b=2,4987608
a*b=5,376*2,499=13,433   a*b=5,37586*2,49876=13,43300
Для продолжения нажмите любую клавишу . . .

```

Рисунок 5.33 – Результат работы программы

3. Переменной вещественного типа  $x$  присваивает значение произведения чисел  $a$ ,  $b$  и  $c$  деленных на их сумму;



результат вывести в формате с плавающей и фиксированной точкой с точностью 4 знака после запятой (рис. 5.34):

```
a=34,0987
b=2,125
c=51,284
x=(34,0987*2,125*51,284)/(34,0987+2,125+51,284)=4,246512E+001
x=(34,0987*2,125*51,284)/(34,0987+2,125+51,284)=42,4651
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.34 – Результат работы программы

4. Запрашивает с клавиатуры три целых числа и выводит следующее сообщение  $[a*(b+c)] = [(b+c)*a]$  (рис. 5.35):

```
a=35
b=60
c=25
[a*(b+c)]=[(b+c)*a]: [35*(60+25)]=[(60+25)*35]=2975
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.35– Результат работы программы

5. Запрашивает с клавиатуры три целых числа, и выводит на экран результат умножения первого числа на  $\pi$  (с точностью 4 знака после запятой), второго на число  $E$  (с точностью 3 знака после запятой) и третьего числа на произведение числа  $\pi$  на  $E$  (с точностью 2 знака после запятой) (рис. 5.36):

```
a=5
b=4
c=3
a*PI: 5*3,1416=15,7080    b*E: 4*2,718=10,873    c*PI*E: 3*3,14*2,72=25,62
Для продолжения нажмите любую клавишу . . .
```

Рисунок 5.36– Результат работы программы

## ПРАКТИЧЕСКАЯ РАБОТА №6. СТРОКИ

**Цель работы:** приобретение навыков алгоритмизации и программирования задач, оперирующих строковыми типами данных:

- ввод и вывод строковых данных;
- обработка строковых данных;
- использование стандартных процедур и функций языка C# для обработки строковых данных.

### ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

За представление строк в C# отвечает класс `System.String`. В коде, для объявления переменной соответствующего типа, предпочтительно использовать следующий вариант написания: `string` – с маленькой буквы. Это ключевое слово языка, используя которое можно объявлять строковые переменные, также как `int` является псевдонимом для `System.Int32`, а `bool` – для `System.Boolean`.

Допустимо объявление строковых переменных через ключевое слово `var` (рис. 6.1):

```
var s2 = "Create by var";  
Console.WriteLine(s2);
```

Рисунок 6.1 – Объявление переменных

Для объединения строк используется оператор «+» (рис. 6.2):

```
string s3 = "Hello, ";  
string s4 = s3 + "John!";  
Console.WriteLine(s4);
```

Рисунок 6.2 – Объединение строк

При работе со `String` следует помнить, что при переопределении значения переменной создается новый экземпляр строковой переменной в памяти. Поэтому, если вам нужно собрать строку из большого количества составляющих, то использование оператора `+` не самый лучший вариант.

В этом случае будет происходить перерасход памяти: при выполнении операции объединения с присваиванием для очень большого количества подстрок, приложение может аварийно завершиться из-за того, что сборщик мусора не будет успевать удалять неиспользуемые объекты, а новые будут продолжать

появляться с большой скоростью. Для решения этой задачи используйте `StringBuilder` (описан ниже).

### Создание и инициализация объекта класса `String`

Существует несколько способов создать объект класса `String` и проинициализировать его. Рассмотрим варианты, которые доступны в C#. Наиболее распространенный способ сделать эту операцию – это присвоить строковое значение переменной без явного вызова конструктора (рис. 6.3):

```
string s5 = "test1";
var s6 = "test2";
```

Рисунок 6.3 – Присваивание значений

Если требуется подготовить строковое значение с использованием набора переменных, то можно воспользоваться статическим методом `Format` класса `String`, либо префиксом `$` (рис. 6.4):

```
int age = 27;
Console.WriteLine(String.Format("Age: {0}", age));
Console.WriteLine("");
Console.WriteLine($"Age: {age}");
Console.WriteLine("");
```

Рисунок 6.4 – Использование `string.Format`

Можно явно вызвать конструктор типа с передачей в него параметров. Самый простой вариант – это передать строку (рис 6.5):

```
string s7 = new string('test3');
```

Рисунок 6.5 – Присваивание значения

В качестве параметра может выступать массив `Char` элементов (рис 6.6):

```
char[] charArray = { 'H', 'e', 'l', 'l', 'o' };
string s8 = new string(charArray);
```

Рисунок 6.6 – Присваивание значения

### Методы и свойства класса `String` Объединение строк.

#### Оператор `+`, методы `Concat` и `Join`

Сцеплять строки между собой можно с помощью оператора `+`, при этом, в результате объединения, будет создан новый объект (рис. 6.7):

```
string s10 = "Area";
string s11 = " 51";
Console.WriteLine("Concat by +: " + s10 + s11);
```

Рисунок 6.7 – Объединение строк

В составе API, который предоставляет System.String, есть метод Concat, который может выполнять ту же работу (рис. 6.8):

```
Console.WriteLine("Concat by Concat(): " + string.Concat(s10, s11));
```

Рисунок 6.8 – Объединение строк

Метод Concat позволяет объединить до четырех строк через прямое перечисление. Если нужно таким образом объединить больше строковых переменных и значений, то используйте оператор +. Полезным свойством Concat является то, что он может принять на вход массив элементов типа String и объединить их (рис. 6.9):

```
string[] sArr1 = { "First ", "Second ", "Third " };
Console.WriteLine(string.Concat(sArr1));
```

Рисунок 6.9 – Объединение строк

Для объединения элементов с указанием разделителя используется метод Join. В предыдущем примере, элементы в массиве sArr1 уже содержали пробел, это не всегда удобно, решим задачу объединения элементов, которые не содержат разделителей, с помощью Join (рис. 6.10):

```
string[] sArr2 = { "First", "Second", "Third" };
Console.WriteLine("Join elements in array by Join() with space: " + string.Join(" ", sArr2));
```

Рисунок 6.10 – Объединение строк

### Поиск и извлечение элементов из строки

Оператор [], методы IndexOf, IndexOfAny, LastIndexOf, LastIndexOfAny, Substring Для получения символа из строки с конкретной позиции (рис. 6.11) можно использовать синтаксис подобный тому, что применяется при работе с массивами – через квадратные скобки []:

```
string s12 = "Hello";
Console.WriteLine("Get element by index s12[3]: " + s12[3]);
```

Рисунок 6.11 – Поиск элемента с определенным индексом

Для решения обратной задачи: поиск индекса первого (последнего) вхождения элемента или строки в данной строке используются методы IndexOf,

IndexOfAny и LastIndexOf, LastIndexOfAny.

В таблице 6.1 перечислены некоторые из предоставляемых System.String вариантов этих методов. Рассмотрим использование методов из таблицы (рис. 6.12) и результаты вывода кода на консоль (рис. 6.13).

Таблица 6.1 – Методы System.String

Метод	Описание
<i>IndexOf(Char)</i>	Возвращает индекс первого вхождения символа.
<i>IndexOf(Char, Int32)</i>	Возвращает индекс первого вхождения символа начиная с заданной позиции.
<i>IndexOf(Char, Int32, Int32)</i>	Возвращает индекс первого вхождения символа начиная с заданной позиции, проверяется указанное количество элементов.
<i>IndexOf(String)</i> <i>IndexOf(String, Int32)</i> <i>IndexOf(String, Int32, Int32)</i>	Назначение методов совпадает с перечисленными выше, но поиск выполняется для строки.
<i>IndexOfAny(Char[])</i> <i>IndexOfAny(Char[], Int32)</i> <i>IndexOfAny(Char[], Int32, Int32)</i>	Назначение методов совпадает с перечисленными выше, но выполняется поиск индекса первого вхождения любого из переданных в массиве элементов.
<i>LastIndexOf(Char   String)</i> <i>LastIndexOf(Char   String, Int32)</i> <i>LastIndexOf(Char   String, Int32, Int32)</i>	Возвращает индекс последнего вхождения символа или строки. Можно задавать индекс, с которого начинать поиск и количество проверяемых позиций. <i>[Char   String]</i> – означает <i>Char</i> или <i>String</i>
<i>LastIndexOfAny(Char[])</i> <i>LastIndexOfAny(Char[], Int32)</i> <i>LastIndexOfAny(Char[], Int32, Int32)</i>	Возвращает индекс последнего вхождения любого из переданных в массиве элементов. Можно задавать индекс с которого начинать поиск и количество проверяемых позиций

```
string s1 = "Hello World!";

// Поиск первого вхождения символа 'r'
Console.WriteLine("Index of \'r\': " + s1.IndexOf('r'));
// Поиск первого вхождения символа 'l' начиная с позиции 4
Console.WriteLine("Index of \'l\', start at 4: " + s1.IndexOf('l', 4));
// Поиск первого вхождения строки "World"
Console.WriteLine("Index of \'World\': " + s1.IndexOf("World"));
// Поиск первого вхождения символа из набора ['o', 'd', ',', '']
Console.WriteLine("Index of pos of any symbol in array: " + s1.IndexOfAny(new char[] { 'o', 'd', ',', ' ' }));
// Поиск последнего вхождения символа 'l'
Console.WriteLine("Last index of \'l\': " + s1.LastIndexOf('l'));
// Поиск последнего вхождения строки "or"
Console.WriteLine("Last index of \'or\': " + s1.LastIndexOf("or"));
// Поиск последнего вхождения символа из набора ['o', 'd', ',', '']
Console.WriteLine("Last index of pos of any symbol in array: " + s1.LastIndexOfAny(new char[] { 'o', 'd', ',', ' ' }));
```

Рисунок 6.12 – Использование методов, для поиска вхождений элементов

```

Index of 'r': 8
Index of 'l', start at 4: 9
Index of "World": 6
Index of pos of any symbol in array: 4
Last index of 'l': 9
Last index of "or": 7
Last index of pos of any symbol in array: 10

```

Рисунок 6.13 – Результат работы программы

Для определения того, содержит ли данная строка указанную подстроку, а также для проверки равенства начала или конца строки заданному значению используйте методы: *Contains*, *StartsWith* и *EndsWith* (таблица 6.2). Рассмотрим использование методов из таблицы (рис. 6.14) и результаты вывода кода на консоль (рис. 6.15).

Таблица 6.2 – Методы System.String

Метод	Описание
<i>Contains(Char)</i> <i>Contains(String)</i>	Возвращает <i>True</i> если строка содержит указанный символ или подстроки.
<i>StartsWith(Char)</i> <i>StartsWith(String)</i>	Возвращает <i>True</i> если строка начинается с заданного символа или подстроки.
<i>EndsWith(Char)</i> <i>EndsWith(String)</i>	Возвращает <i>True</i> если строка заканчивается на заданный символ или подстроку.

```

Console.WriteLine("Contains \"World\"? " + s1.Contains("World")); // True
Console.WriteLine("Starts with \"He\"? " + s1.StartsWith("He")); // True
Console.WriteLine("Ends with \"ld\"? " + s1.EndsWith("ld")); // False

```

Рисунок 6.13 – Использование методов, для поиска вхождений элементов

```

Contains "World"? True
Starts with "He"? True
Ends with "ld"? False

```

Рисунок 6.14 – Результат работы программы

Задачу извлечения подстроки из данной строки решает метод *SubString*. (таблица 6.3). Рассмотрим использование методов из таблицы (рис. 6.15) и результаты вывода кода на консоль (рис. 6.16).

Таблица 6.3 – Методы System.String



Метод	Описание
<i>Substring(Int32)</i>	Возвращает подстроку начиная с указанной позиции и до конца исходной строки.
<i>Substring(Int32, Int32)</i>	Возвращает подстроку начиная с указанной позиции с заданной длины.

```
Console.WriteLine("Substring start at pos 7: " + s1.Substring(7)); // World!
Console.WriteLine("Substring start at pos 7 (4 chars): " + s1.Substring(7, 4)); // Worl
```

Рисунок 6.15 – Использование методов извлечения подстроки

```
Substring start at pos 7: orld!
Substring start at pos 7 (4 chars): orld
```

Рисунок 6.14 – Результат работы программы

### Сравнение строк

Для сравнения строк можно использовать оператор сравнения `==`, при этом будут сравниваться значения строковых переменных, а не их ссылки, как это делается для других ссылочных типов (рис. 6.15).

```
string t1 = "John";
string t2 = "John";
string t3 = "Mary";
Console.WriteLine("t1 == t2: " + (t1 == t2)); // True
Console.WriteLine("t1 != t2: " + (t1 != t2)); // False
Console.WriteLine("t1 == t3: " + (t1 == t3)); // False
```

Рисунок 6.15 – Сравнение строк

Для сравнения также можно использовать метод `Equals`, но это менее удобный вариант (рис 6.16):

```
Console.WriteLine("Equals method: t1.Equals(t2)" + t1.Equals(t2)); // True
Console.WriteLine("Equals method: t1.Equals(t3)" + t1.Equals(t3)); // False
```

Рисунок 6.16 – Сравнение строк с помощью метода `Equals`

### Модификация строк

Класс `String` предоставляет довольно большое количество инструментов для изменения строк.

Вставка строки в исходную в заданную позицию осуществляется с помощью метода `Insert` (рис 6.17):

```
Console.WriteLine("Insert: " + "26".Insert(1, "[4]")); // 2[4]6
```

Рисунок 6.17 – Вставка строки

Для приведения строки к заданной длине с выравниванием по левому (правому) краю с заполнением недостающих символов пробелами используются методы `PadLeft` и `PadRight` (рис 6.18):

```
Console.WriteLine("PadLeft: ");
Console.WriteLine("some string".PadLeft(15)); // " some string"
Console.WriteLine("some string".PadLeft(15, '*')); // "*****some string"
Console.WriteLine("PadRight: ");
Console.WriteLine("some string".PadRight(15)); // "some string "
Console.WriteLine("some string".PadRight(15, '*')); // "some string*****"
```

Рисунок 6.18 – Приведение строки к заданной длине

Метод `Remove` (рис. 6.19) удаляет подстроку из исходной строки. Возможны два варианта использования (таблица 6.4).

Таблица 6.4 – Методы `System.String`

Метод	Описание
<code>Remove(Int32)</code>	Удаляет все символы начиная с заданного и до конца строки.
<code>Remove(Int32, Int32)</code>	Удаляет с указанной позиции заданное число символов.

```
Console.WriteLine("Remove demo1: " + "Hello".Remove(2));
Console.WriteLine("Remove demo2: " + "Hello".Remove(2, 2));
```

Рисунок 6.19 – Удаление подстроки

Замена элементов строки производится с помощью метода `Replace`. Наиболее часто используемые варианты – это замена символа на символ и строки на подстроку (рис 6.20):

```
Console.WriteLine("Hello, World!".Replace('!', '.')); // Hello, World.
Console.WriteLine("Hello, World!".Replace("World", "John")); // Hello, John!
```

Рисунок 6.20 – Замена элементов строки

Для преобразования строки к верхнему регистру используйте метод `ToUpper()`, к нижнему – `ToLower()` (рис. 6.21):

```
Console.WriteLine("Hello, World!".ToUpper()); // HELLO, WORLD!
Console.WriteLine("Hello, World!".ToLower()); // hello, world!
```

Рисунок 6.21 – Преобразование строки к верхнему и нижнему регистру

За удаление начальных и конечных символов отвечают методы, начинающиеся на `Trim` (рис 6.22, таблица 6.5).

Таблица 6.5 – Методы `System.String`



Метод	Описание
<i>Trim()</i>	Удаляет символы пробелы из начала и конца строки.
<i>Trim(Char)</i>	Удаляет экземпляры символа из начала и конца строки.
<i>Trim(Char[])</i>	Удаляет экземпляры символов из начала и конца строки.
<i>TrimStart()</i> <i>TrimStart(Char)</i> <i>TrimStart(Char[])</i>	Удаляет экземпляры символов из начала строки.
<i>TrimEnd()</i> <i>TrimEnd(Char)</i> <i>TrimEnd(Char[])</i>	Удаляет экземпляры символов из конца строки.

```
Console.WriteLine("***hello---".Trim(new char[] { '*', '-' })); // "hello"
Console.WriteLine(" hello ".TrimStart()); // "hello "
Console.WriteLine(" hello ".TrimEnd()); // " hello"
```

Рисунок 6.22 – Удаление начальных и конечных символов строки

### Методы и свойства общего назначения

Рассмотрим некоторые из полезных методов и свойств, которые не вошли в приведенные выше группы.

`System.Length` – возвращает длину строки (рис 6.23):

```
Console.WriteLine("Hello".Length); // 5
```

Рисунок 6.23 – Длина строки

`System.Split()` – разделяет заданную строку на подстроки, в качестве разделителя используется указанный через параметр символ (рис 6.24):

`System.Empty` – возвращает пустую строку.

```
foreach (var s in "1 2 3".Split(' '))
    Console.WriteLine(s);
foreach (var s in "1 2 3-4-5-6".Split(new char[] { ' ', '-' }))
    Console.WriteLine(s);
```

Рисунок 6.24 – Разделение строки на подстроки

## ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №6

### Вариант 1

1. Создайте из четных символов строки `str1` строку `str2` и из нечетных символов - строку `str3`.
2. В заданном тексте удалить часть текста, заключенную в скобки (вместе со скобками).

### Вариант 2

1. Написать программу, которая подсчитывает, сколько содержится цифр в строке.
2. Определить сколько раз в тексте встречается заданное слово (слова разделены пробелами).

### **Вариант 3**

1. Написать программу, которая подсчитывает, сколько содержится цифр в строке.
2. В тексте вставить между словами вместо одного пробела запятую и пробел.

### **Вариант 4**

1. Вводится строка из букв и цифр. Построить новую строку только из цифр (порядок не меняется).
2. Определить, какой процент слов в тексте начинается на букву К. Слова разделены пробелами.

### **Вариант 5**

1. В строке, состоящей из букв и цифр необходимо оставить только один экземпляр каждого встречающегося символа.
2. Написать программу, исключаящую из символьной строки все цифры.

### **Вариант 6**

1. Необходимо перевернуть строку, т.е. последние символы должны стать первыми, а первые последними.
2. Определить, содержит ли данный текст символы, отличные от букв и пробела.

### **Вариант 7**

1. С клавиатуры вводятся две строки. Найти количество вхождений одной (являющейся подстрокой) в другую.
2. Дано предложение. Удалить из него буквы «р» и «с» (создать процедуру, удаляющую из строки заданный символ).

### **Вариант 8**

1. Преобразовать три введенные строки, чтобы после каждой цифры следовал символ «!» (создать процедуру, вставляющую пробел после каждой цифры в строке)
2. Дано предложение. Все пробелы в нем заменить на символ "\_" (создать соответствующую процедуру).

**Вариант 9**

1. Отсортировать массив строк по алфавиту. Учитывать только первый символ каждой строки.
2. Введенную строку букв и цифр преобразовать так, чтобы после каждой цифры следовал пробел.

**Вариант 10**

1. Дана строка, состоящая из слов, разделенных пробелами и знаками препинания. Определить длину самого короткого слова.
2. Изменить введенную строку так, чтобы каждая из цифр увеличилась на 1 (9 заменить 0).

## ПРАКТИЧЕСКАЯ РАБОТА №7. РАБОТА С МАССИВАМИ

**Цель работы:** Получение практических навыков в работе с одномерными массивами.

### ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

**Массив** – это совокупность переменных одного типа, к которым обращаются с помощью общего имени. Доступ к отдельному элементу массива может осуществляться с помощью индекса. В языке C# все массивы состоят из соприкасающихся участков памяти. Наименьший адрес соответствует первому элементу, наибольший адрес соответствует последнему элементу. Массивы могут иметь одну или несколько размерностей.

**Одномерный массив** – это фиксированное количество элементов одного и того же типа, объединенных общим именем, где каждый элемент имеет свой номер. Нумерация элементов массива в C# начинается с **нуля**, то есть, если массив состоит из 10 элементов, то его элементы будут иметь следующие номера: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Одномерный массив в C# реализуется как объект, по этому его создание представляет собой двухступенчатый процесс. Сначала объявляется ссылочная переменная на массив, затем выделяется память под требуемое количество элементов базового типа, и ссылочной переменной присваивается адрес нулевого элемента в массиве. Базовый тип определяет тип данных каждого элемента массива. Количество элементов, которые будут храниться в массиве, определяется размер массива.

В общем случае процесс объявления переменной типа массив, и выделение необходимого объема памяти может быть разделено. Кроме того, на этапе объявления массива можно произвести его инициализацию. Поэтому для объявления одномерного массива может использоваться одна из следующих форм записи, записанных в таблице 7.1.

Таблица 7.1. Формы записи массивов

Форма записи	Пояснения
базовый_тип [] имя__массива; Например: <code>int [] a;</code>	Описана ссылка на одномерный массив, которая в дальнейшем может быть использована: 1) для адресации на уже существующий массив; 2) передачи массива в метод в качестве параметра 3) отсроченного выделения памяти под элементы массива.
базовый_тип [] имя__массива = new базовый_тип [размер]; Например: <code>int []a=new int [10];</code>	Объявлен одномерный массив заданного типа и выделена память под одномерный массив указанной размерности. Адрес данной области памяти записан в ссылочную переменную. Элементы массива равны нулю.
базовый_тип [] имя массива={список инициализации}; Например: <code>int []a={0, 1, 2, 3};</code>	Выделена память под одномерный массив размерность которого соответствует количеству элементов в списке инициализации. Адрес этой области памяти записан в ссылочную переменную. Значение элементов массива соответствует списку инициализации.

Обращения к элементам массива происходят с помощью индекса, для этого нужно указать имя массива и в квадратных скобках его номер. Например: `a[0]`, `b[10]`, `c[i]`.

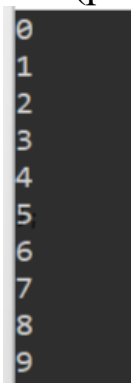
В C# индексация массивов начинается с нуля. Так как массив представляет собой набор элементов, объединенных общим именем, то обработка массива обычно производится в цикле. Рассмотрим несколько простых примеров работы с одномерными массивами.

**Пример 1.** Вывод элементов массива на консоль (рис 7.1).

```
int[] myArray = { 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 };
int i;
for (i = 0; i < 10; ++i)
    Console.WriteLine(myArray[i]);
```

Рисунок 7.1 – Вывод массива на консоль

Результат работы программы (рис 7.2)



```
0
1
2
3
4
5
6
7
8
9
```

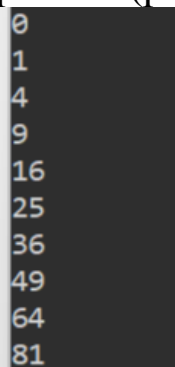
Рисунок 7.2 – Вывод массива на консоль

**Пример 2.** Заполнение массива квадратами его индексов и вывод массива на консоль (рис 7.3).

```
int[] myArray = new int[10];
int i;
for (i = 0; i < 10; i++)
    myArray[i] = i * i;
for (i = 0; i < 10; i++)
    Console.WriteLine(myArray[i]);
```

Рисунок 7.3 – Заполнение массива квадратами его индексов

Результат работы программы (рис 7.4).



```
0
1
4
9
16
25
36
49
64
81
```

Рисунок 7.4 – Вывод массива на консоль

### Реализация некоторых алгоритмов в массиве

#### 1. Поиск максимального/минимального элемента массива.

Если в массиве элементов больше одного, то за максимальный в начале принимаем первый элемент массива. Далее находить максимум будем последовательно, сравнивая текущий элемент с максимумом, найденным на предыдущем шаге. Если текущий элемент больше, то значение максимума, найденное на предыдущем шаге, нужно обновить.

Рассмотрим алгоритм поиска в виде блок-схемы (рис 7.5).

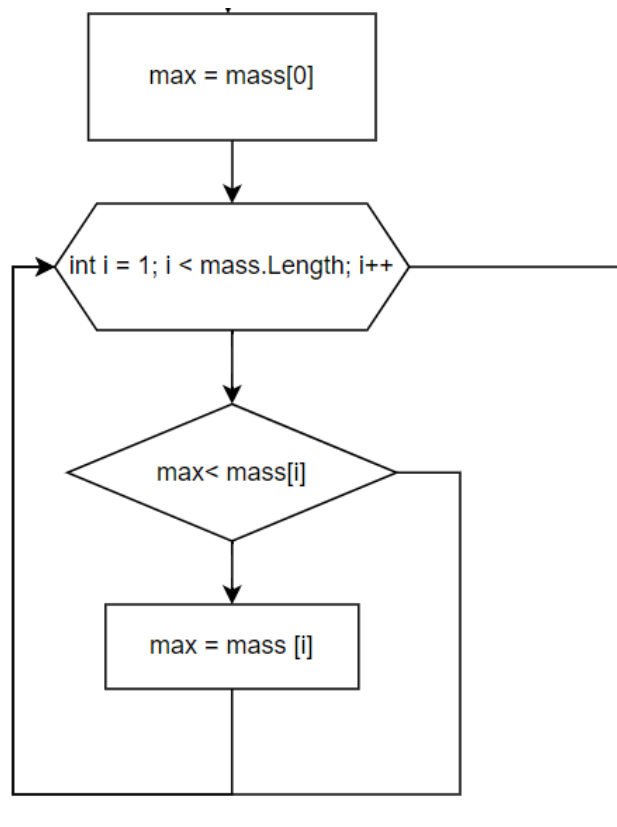


Рисунок 7.5 – Поиск максимального/минимального элемента массива

## 2. Поиск наиболее повторяющегося элемента массива.

Для реализации данного алгоритма необходимо использовать два вложенных цикла для анализа массива чисел. Внешний цикл проходится по каждому элементу массива, представляя собой текущее число для анализа. Внутренний цикл затем сканирует оставшиеся элементы массива, подсчитывая количество повторений текущего числа.

Для каждого числа внутренний цикл считает, сколько раз это число встречается в массиве. Если количество повторений текущего числа больше, чем количество повторений числа, которое ранее было наиболее часто встречающимся, то обновляются переменные, отслеживающие наиболее часто встречающееся число и его частоту.

Рассмотрим алгоритм поиска в виде блок-схемы (рис 7.6).

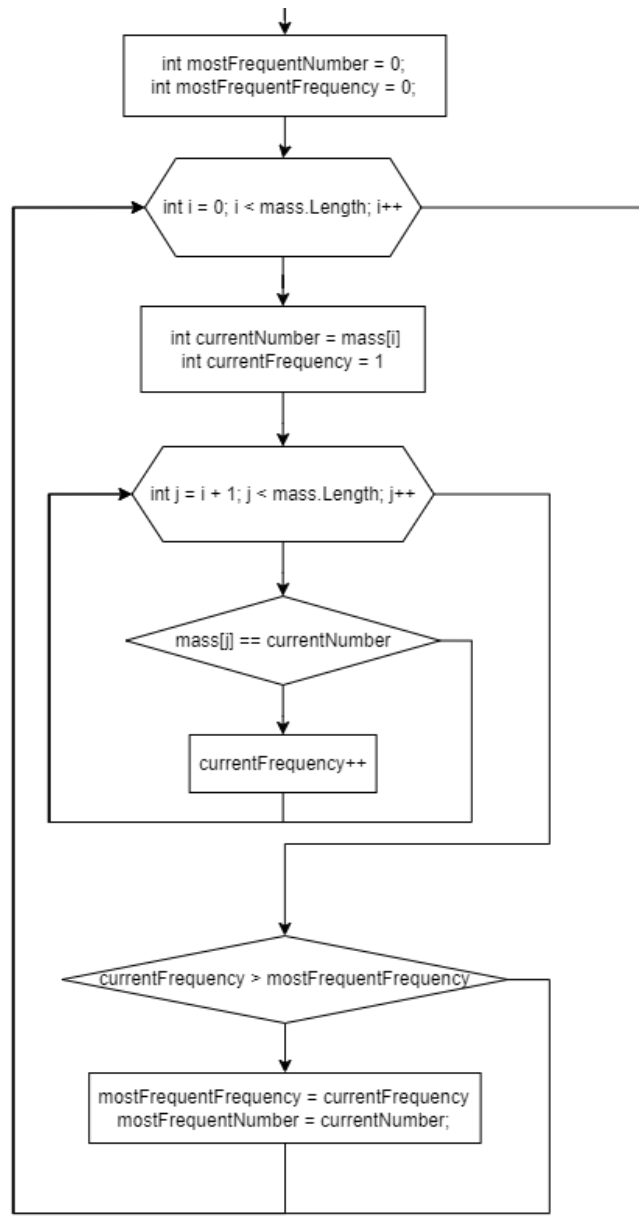


Рисунок 7.6 – Поиск наиболее повторяющегося элемента массива

## ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №7.

### Вариант 1

В одномерном массиве, состоящем из **n** вещественных элементов, вычислить:

- 1) сумму отрицательных элементов массива;
- 2) произведение элементов массива, расположенных между максимальным и минимальным элементами.
- 3) количество и сумму элементов, кратных заданному числу **K**.



Числа  $n$  и  $K$  – задает пользователь.

### Вариант 2

В одномерном массиве, состоящем из  $n$  вещественных элементов, вычислить:

- 1) сумму положительных элементов массива;
- 2) произведение элементов массива, расположенных между максимальным по модулю и минимальным по модулю элементами.
- 3) в массиве есть нулевые элементы, создайте новый массив, который будет содержать индексы нулевых элементов.

Размерность массива  $n$  – задается пользователем.

### Вариант 3

В одномерном массиве, состоящем из  $n$  целых элементов, вычислить:

- 1) произведение элементов массива с четными номерами;
- 2) сумму элементов массива, расположенных между первым и последним нулевыми элементами.
- 3) из четных элементов массива создайте новый массив.

Размерность массива  $n$  – задается пользователем.

### Вариант 4

В одномерном массиве, состоящем из  $n$  вещественных элементов, вычислить:

- 1) сумму элементов массива с нечетными номерами;
- 2) сумму элементов массива, расположенных между первым и последним отрицательными элементами.
- 3) сжать массив, удалив из него все элементы, модуль которых **не** превышает 1, освободившиеся элементы заполнить нулями.

Размерность массива  $n$  – задается пользователем.

### Вариант 5

В одномерном массиве, состоящем из  $n$  вещественных

элементов, вычислить:

- 1) максимальный элемент массива;
- 2) сумму элементов массива, расположенных до последнего положительного элемента.
- 3) сжать массив, удалив из него все элементы, модуль которых находится в интервале  $[a, b]$ , освободившиеся элементы заполнить нулями.

Размерность массива  $n$  и интервал  $a; b$  – задается пользователем.

### Вариант 6

В одномерном массиве, состоящем из  $n$  вещественных элементов, вычислить:

- 1) минимальный элемент массива;
- 2) сумму элементов массива, расположенных между первым и последним положительными элементами.
- 3) заменить все элементы массива, большие заданного числа  $K$ , этим числом и подсчитать количество замен.

Размерность массива  $n$  и число  $K$  – задается пользователем.

### Вариант 7

В одномерном массиве, состоящем из  $n$  целых элементов, вычислить:

- 1) номер максимального элемента массива.
- 2) произведение элементов массива, расположенных между первым и вторым нулевыми элементами.
- 3) поменять местами наибольший и наименьший элементы заданного массива.

Размерность массива  $n$  – задается пользователем.

### Вариант 8

В одномерном массиве, состоящем из  $n$  вещественных элементов, вычислить:

- 1) номер минимального элемента массива;

- 2) сумму элементов массива, расположенных между первым и вторым отрицательными элементами.
- 3) в массиве найти наиболее часто встречающееся число.

Размерность массива  $n$  – задается пользователем.

### **Вариант 9**

В одномерном массиве, состоящем из  $n$  вещественных элементов, вычислить:

- 1) максимальный по модулю элемент массива.
- 2) сумму элементов массива, расположенных между первым и вторым положительными элементами.
- 3) сумму элементов, расположенных между максимальным и минимальным элементами массива, включая оба эти числа.

Размерность массива  $n$  – задается пользователем.

### **Вариант 10**

В одномерном массиве, состоящем из  $n$  целых элементов, вычислить:

- 1) минимальный по модулю элемент массива.
- 2) сумму модулей элементов массива, расположенных после первого элемента, равного нулю.
- 3) произведения всех пар соседних чисел.

Размерность массива  $n$  – задается пользователем.

## ПРАКТИЧЕСКАЯ РАБОТА №8. ПРОГРАММИРОВАНИЕ МОДУЛЯ. ОРГАНИЗАЦИЯ ПРОЦЕДУР И ФУНКЦИЙ

**Цель работы:** Изучение основ программирования стандартных модулей, организации процедур и функций.

### ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

**Подпрограмма** – это отдельная часть программы, имеющая имя и решающая свою отдельную задачу. Располагается подпрограмма в начале основной программы и может быть запущена (вызвана) из основной программы по указанию имени.

Использование подпрограмм позволяет избежать дублирования кода, в случае если необходимо один и тот же код писать в разных местах программы. Библиотеки, которые импортируются в программу (например, System) состоят из подпрограмм, которые уже были кем-то составлены. Программистам не нужно думать о том, какие алгоритмы в них реализованы, они просто применяют их, задумываясь только о том, что именно они делают. Это большая экономия времени. Нет нужды писать алгоритм, который уже был кем-то написан.

Каждая подпрограмма должна решать только одну задачу, либо что-то вычислять, либо выводить какие-либо данные, либо делать что-то еще.

Подпрограммы, или методы, бывают двух типов - **функции** (те, которые возвращают результат работы) и **процедуры** (те, которые не возвращают).

Процедуры и функции связываются с классом, они обеспечивают функциональность данных класса и называются методами класса. Главную роль в программной системе играют данные, а функции лишь служат данным. В С# процедуры и функции существуют только как методы некоторого класса, они не существуют вне класса. В данном контексте понятие класс распространяется и на все его частные случаи - структуры, интерфейсы, делегаты.

Функция отличается от процедуры двумя особенностями. Она всегда вычисляет некоторое значение, возвращаемое в качестве результата функции и вызывается в выражениях.

Процедура С# имеет свои особенности:

- Она возвращает формальный результат `void`, указывающий на отсутствие результата;
- Вызов процедуры является оператором языка;
- И она имеет входные и выходные аргументы, причем выходных аргументов - ее результатов - может быть достаточно много.

Предположим, что нам нужно выводить на экран строку "Error" каждый раз, когда в коде может возникнуть ошибка по вине пользователя (например, когда он вводит неверные данные).

Это можно сделать, написав оператор:

```
Console.WriteLine("Error");
```

А теперь представим, что такую строчку нужно вставить во многих местах программы. Конечно, можно просто везде ее написать. Но это решение имеет два недостатка.

- 1) Данная строка будет храниться в памяти много раз;
- 2) Если мы захотим изменить вывод при ошибке, то придется менять эту строку по всей программе, что достаточно неудобно.

Для таких случаев и нужны методы и процедуры.

Программа с процедурой может выглядеть следующим образом:

```
using System;
class Program
{
    static void PrintError ()
    {
        Console.WriteLine("Error");
    }
    static void Main ()
    {
        PrintError ();
    }
}
```

Процедура начинается со слова `void`. После имени процедуры записаны пустые скобки.

Все операторы, которые выполняются в процедуре, записываются с отступом.

Модификатор `static` означает, что данное поле, метод или свойство будет принадлежать не каждому объекту класса, а всем им вместе.

Методы и процедуры записываются до главного метода `Main()`.

Чтобы обратиться к процедуре, в основной программе необходимо вызвать ее по имени и не забыть написать скобки.

Вызывать процедуру в программе можно сколько угодно раз.

А теперь представим, что нам необходимо в ответ на ошибку пользователя вывести разные сообщения, в зависимости от того, какую именно ошибку он сделал.

В этом случае можно для каждой ошибки написать свою процедуру:

```
void printErrorZero ()
{
    Console.WriteLine("Error. Division by zero!");
}
void printErrorInput ()
{
    Console.WriteLine("Error in input!");
}
```

А если возможных ошибок будет намного больше? Тогда такое решение нам не подойдет.

Надо научиться управлять процедурой, указывая ей, какое сообщение на ошибку нужно вывести.

Для этого нам понадобятся параметры, которые мы будем записывать в круглых скобках, после имени процедуры.

```
void printError (string s)
{
    Console.WriteLine(s);
}
```

В данной процедуре `s` — это параметр - специальная переменная, которая позволяет управлять процедурой.

Параметр — это переменная, от значения которой зависит работа подпрограммы. Имена параметров перечисляются через запятую в заголовке подпрограммы. Перед параметром записывается его тип.

Теперь при вызове процедуры нужно в скобках указывать фактическое значение, которое будет присвоено параметру (переменной `s`) внутри нашей процедуры `printError("Error! Division by zero!")`. Такое значение называется аргументом.

Аргумент – это значение параметра, которое передается подпрограмме при ее вызове.

Аргументом может быть не только постоянное значение, но и переменная или арифметическое выражение.

## **ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №8**

### **Вариант 1**

1. Вводится натуральное число. Написать функцию, вычисляющий сумму четных цифр, входящих в состав этого числа.

2. Написать функцию, выводящую элементы числового массива, которые больше, чем элементы, стоящие перед ними. Например, дан массив `[3, 9, 8, 4, 5, 1]`. Следует вывести числа 9 и 5, так как перед ними стоят соответственно числа 3 и 4, которые меньше их.

### **Вариант 2**

1. Написать функцию, которая делит все элементы массива на значение наибольшего элемента этого массива.

2. Написать функцию, которая возвращает среднее арифметическое двух переданных ей аргументов (параметров).

### **Вариант 3**

1. Написать функцию нахождения по заданной длине стороны треугольника и величинам двух прилежащих углов длин остальных сторон треугольника и величины третьего угла.

2. Написать функцию вычисления  $f(x)$  по формуле от заданного  $x$ . Класс `Math` использовать нельзя:

$$f(x) = x^2 \quad \text{при } -2 \leq x < 2;$$

$$f(x) = x^2 + 4x + 5 \quad \text{при } x \geq 2;$$

$$f(x) = \operatorname{tg}(2x^5) \quad \text{при } x < -2.$$

### **Вариант 4**

1. Написать функцию для обмена значений двух переменных.

2. Составить программу вычисления данного выражения:  $y = (x^6 * (x-5)^3) / (2*x+1)^5$ . Возведение выражений в степень с натуральным показателем оформить в виде функции, как нахождение произведения одинаковых множителей. Не использовать стандартной математической функции вычисления степени (т.е. написать самостоятельно).

### **Вариант 5**

1. С помощью подпрограммы заполнить матрицы случайными числами. Написать функцию, вычисляющую сумму двух матриц. Вывести на экран две исходные матрицы и их сумму (используя функцию).

2. Написать функцию, преобразующую число, введенное пользователем, в десятичной системе счисления в число, выраженное в шестнадцатеричной системе счисления. Результат вывести на экран.

### **Вариант 6**

1. Написать функцию, вычисляющую корни квадратного уравнения.

2. Написать функцию, которая преобразует двоичное число, введенное пользователем программы, в десятичное число. Результат вывести на экран.

### **Вариант 7**

1. Создать функцию, для поиска минимального значения из заданного массива, заполненного рандомными числами.

2. Написать функцию, которая принимает переменное количество целых чисел, возводит их в квадрат, и возвращает массив значений.

### **Вариант 8**

1. Написать функцию подсчета количества часов, минут и секунд в введенном количестве суток.

2. Составить функцию для определения лежит ли точка  $(x_3; y_3)$ , на прямой проходящей через точки  $(x_1; y_1)$ ,  $C(x_2; y_2)$ .

### **Вариант 9**



1. Даны координаты вершин треугольника ABC:  $A(x_1; y_1)$ ,  $B(x_2; y_2)$ ,  $C(x_3; y_3)$ . Написать функцию, которая будет определять ли треугольник равнобедренным, прямоугольным, правильным.

2. Пользователь вводит с клавиатуры четыре вещественных числа. Необходимо написать функцию, которая будет находить максимальное из них и функцию, которая будет выводить на экран максимальное число.

### **Вариант 10**

1. Составить функцию для вычисления среднего арифметического числа одномерного массива, размер массива задается пользователем.

2. Написать функцию возведения в степень  $N$  каждого элемента из массива  $A$ . Результат вывести на экран.

## ПРАКТИЧЕСКАЯ РАБОТА №9. ОРГАНИЗАЦИЯ ДОСТУПА К ОБЪЕКТАМ ТАБЛИЧНОГО ПРОЦЕССОРА

**Цель работы:** изучение способов обращения к основным объектам табличного процессора.

### ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Термин **Объекты Excel** (понимаемый в широком смысле, как объектная модель Excel) включает в себя элементы, из которых состоит любая рабочая книга Excel. Это, например, рабочие листы (**Worksheets**), строки (**Rows**), столбцы (**Columns**), диапазоны ячеек (**Ranges**) и сама рабочая книга Excel (**Workbook**) в том числе. Каждый объект Excel имеет набор свойств, которые являются его неотъемлемой частью.

Например, объект **Worksheet** (рабочий лист) имеет свойства **Name** (имя), **Protection** (защита), **Visible** (видимость), **Scroll Area** (область прокрутки) и так далее. Таким образом, если в процессе выполнения макроса требуется скрыть рабочий лист, то достаточно изменить свойство **Visible** этого листа.

В Excel VBA существует особый тип объектов – **коллекция**. Как можно догадаться из названия, коллекция ссылается на группу (или коллекцию) объектов Excel. Например, коллекция **Rows** – это объект, содержащий все строки рабочего листа.

Доступ ко всем основным объектам Excel может быть осуществлён (прямо или косвенно) через объект **Workbooks**, который является коллекцией всех открытых в данный момент рабочих книг. Каждая рабочая книга содержит объект **Sheets** – коллекция, которая включает в себя все рабочие листы и листы с диаграммами рабочей книги. Каждый объект **Worksheet** состоит из коллекции **Rows** – в неё входят все строки рабочего листа, и коллекции **Columns** – все столбцы рабочего листа, и так далее. В следующей таблице перечислены некоторые наиболее часто используемые объекты Excel.

Таблица 9.1 – Объекты Excel

Объект	Описание
Application	Приложение Excel.

Workbooks	<p>Коллекция всех открытых в данный момент рабочих книг в текущем приложении Excel. Доступ к какой-то конкретной рабочей книге может быть осуществлён через объект <b>Workbooks</b> при помощи числового индекса рабочей книги или её имени, например, <b>Workbooks (1)</b> или <b>Workbooks («Книга1»)</b>.</p>
Workbook	<p>Объект <b>Workbook</b> – это рабочая книга. Доступ к ней может быть выполнен через коллекцию <b>Workbooks</b> при помощи числового индекса или имени рабочей книги (см. выше). Для доступа к активной в данный момент рабочей книге можно использовать <b>ActiveWorkbook</b>.</p> <p>Из объекта <b>Workbook</b> можно получить доступ к объекту <b>Sheets</b>, который является коллекцией всех листов рабочей книги (рабочие листы и диаграммы), а также к объекту <b>Worksheets</b>, который представляет из себя коллекцию всех рабочих листов книги Excel.</p>
Sheets	<p>Объект <b>Sheets</b>– это коллекция всех листов рабочей книги. Это могут быть как рабочие листы, так и диаграммы на отдельном листе. Доступ к отдельному листу из коллекции <b>Sheets</b> можно получить при помощи числового индекса листа или его имени, например, <b>Sheets (1)</b> или <b>Sheets («Лист1»)</b>.</p>
Worksheets	<p>Объект <b>Worksheets</b> – это коллекция всех рабочих листов в рабочей книге (то есть, все листы, кроме диаграмм на отдельном листе). Доступ к отдельному рабочему листу из коллекции <b>Worksheets</b> можно получить при помощи числового индекса рабочего листа или его имени, например, <b>Worksheets (1)</b> или <b>Worksheets («Лист1»)</b>.</p>
Worksheet	<p>Объект <b>Worksheet</b> – это отдельный рабочий лист книги Excel. Доступ к нему можно получить при помощи числового индекса рабочего листа или его имени (см. выше).</p> <p>Кроме этого, Вы можете использовать <b>ActiveSheet</b> для доступа к активному в данный момент рабочему листу. Из объекта <b>Worksheet</b> можно получить доступ к объектам <b>Rows</b> и <b>Columns</b>, которые являются коллекцией объектов <b>Range</b>, ссылающихся на строки и столбцы рабочего листа. А также можно получить доступ к отдельной ячейке или к любому диапазону смежных ячеек на рабочем листе.</p>

Rows	Объект <b>Rows</b> – это коллекция всех строк рабочего листа. Объект <b>Range</b> , состоящий из отдельной строки рабочего листа, может быть доступен по номеру этой строки, например, <b>Rows (1)</b> .
Columns	Объект <b>Columns</b> – это коллекция всех столбцов рабочего листа. Объект <b>Range</b> , состоящий из отдельного столбца рабочего листа, может быть доступен по номеру этого столбца, например, <b>Columns (1)</b> .
Range	<p>Объект <b>Range</b> – это любое количество смежных ячеек на рабочем листе. Это может быть одна ячейка или все ячейки листа.</p> <p>Доступ к диапазону, состоящему из единственной ячейки, может быть осуществлён через объект <b>Worksheet</b> при помощи свойства <b>Cells</b>, например, <b>Worksheet.Cells(1,1)</b>.</p> <p>По-другому ссылку на диапазон можно записать, указав адреса начальной и конечной ячеек. Их можно записать через двоеточие или через запятую.</p> <p>Например, <b>Worksheet.Range(«A1:B10»)</b> или <b>Worksheet.Range («A1», «B10»)</b> или <b>Worksheet.Range (Cells(1,1), Cells(10,2))</b>.</p> <p>Обратите внимание, если в адресе <b>Range</b> вторая ячейка не указана (например, <b>Worksheet.Range(«A1»)</b> или <b>Worksheet.Range (Cells(1,1))</b>, то будет выбран диапазон, состоящий из единственной ячейки.</p>

Приведённая выше таблица показывает, как выполняется доступ к объектам Excel через родительские объекты. Например, ссылку на диапазон ячеек можно записать вот так:

**Workbooks("Книга1").Worksheets("Лист1").Range("A1:B10")**

### **Присваивание объекта переменной**

В Excel VBA объект может быть присвоен переменной при помощи ключевого слова **Set**:

**Dim DataWb As Workbook**

**Set DataWb = Workbooks("Книга1.xlsx")**

### **Активный объект**

В любой момент времени в Excel есть активный объект **Workbook** – это рабочая книга, открытая в этот момент. Точно

так же существует активный объект **Worksheet**, активный объект **Range** и так далее.

Сослаться на активный объект **Workbook** или **Sheet** в коде VBA можно как на **ActiveWorkbook** или **ActiveSheet**, а на активный объект **Range** – как на **Selection**.

Если в коде VBA записана ссылка на рабочий лист, без указания к какой именно рабочей книге он относится, то Excel по умолчанию обращается к активной рабочей книге. Точно так же, если сослаться на диапазон, не указывая определённую рабочую книгу или лист, то Excel по умолчанию обратится к активному рабочему листу в активной рабочей книге.

Таким образом, чтобы сослаться на диапазон **A1:B10** на активном рабочем листе активной книги, можно записать просто: `Range("A1:B10")`

### Смена активного объекта

Если в процессе выполнения программы требуется сделать активной другую рабочую книгу, другой рабочий лист, диапазон и так далее, то для этого нужно использовать методы **Activate** или **Select** вот таким образом:

```
Sub ActivateAndSelect()
    Workbooks("Книга2").Activate
    Worksheets("Лист2").Select
    Worksheets("Лист2").Range("A1:B10").Select
    Worksheets("Лист2").Range("A5").Activate
End Sub
```

Методы объектов, в том числе использованные только что методы **Activate** или **Select**, далее будут рассмотрены более подробно.

### Свойства объектов

Каждый объект VBA имеет заданные для него свойства. Например, объект **Workbook** имеет свойства **Name** (имя), **RevisionNumber** (количество сохранений), **Sheets** (листы) и множество других. Чтобы получить доступ к свойствам объекта, нужно записать имя объекта, затем точку и далее имя свойства. Например, имя активной рабочей книги может быть доступно вот так: **ActiveWorkbook.Name**. Таким образом, чтобы присвоить

переменной **wbName** имя активной рабочей книги, можно использовать вот такой код:

```
Dim wbName As String wbName = ActiveWorkbook.Name
```

Ранее мы показали, как объект **Workbook** может быть использован для доступа к объекту **Worksheet** при помощи такой команды:

```
Workbooks("Книга1").Worksheets("Лист1")
```

Это возможно потому, что коллекция **Worksheets** является свойством объекта **Workbook**.

Некоторые свойства объекта доступны только для чтения, то есть их значения пользователь изменять не может. В то же время существуют свойства, которым можно присваивать различные значения. Например, чтобы изменить название активного листа на «**Мой рабочий лист**», достаточно присвоить это имя свойству **Name** активного листа, вот так:

```
ActiveSheet.Name = "Мой рабочий лист"
```

### Методы объектов

Объекты VBA имеют методы для выполнения определённых действий. **Методы объекта** – это процедуры, привязанные к объектам определённого типа. Например, объект **Workbook** имеет методы **Activate**, **Close**, **Save** и ещё множество других.

Для того, чтобы вызвать метод объекта, нужно записать имя объекта, точку и имя метода. Например, чтобы сохранить активную рабочую книгу, можно использовать вот такую строку кода:

```
ActiveWorkbook.Save
```

Как и другие процедуры, методы могут иметь аргументы, которые передаются методу при его вызове. Например, метод **Close** объекта **Workbook** имеет три необязательных аргумента, которые определяют, должна ли быть сохранена рабочая книга перед закрытием и тому подобное.

Чтобы передать методу аргументы, необходимо записать после вызова метода значения этих аргументов через запятую. Например, если нужно сохранить активную рабочую книгу как файл **.csv** с именем «Книга2», то нужно вызвать метод **SaveAs**

объекта **Workbook** и передать аргументу **Filename** значение **Книга2**, а аргументу **FileFormat** – значение **xlCSV**:

ActiveWorkbook.SaveAs "Книга2", xlCSV

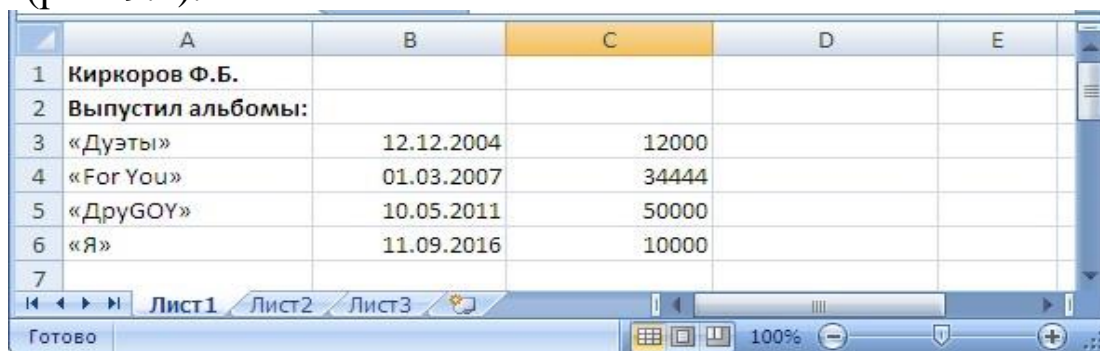
Чтобы сделать код более читаемым, при вызове метода можно использовать именованные аргументы. В этом случае сначала записывают имя аргумента, затем оператор присваивания «:=» и после него указывают значение. Таким образом, приведённый выше пример вызова метода **SaveAs** объекта **Workbook** можно записать по-другому:

ActiveWorkbook.SaveAs Filename:="Книга2", [FileFormat]:=xlCSV

В окне **Object Browser** редактора Visual Basic показан список всех доступных объектов, их свойств и методов. Чтобы открыть этот список, запустите редактор Visual Basic и нажмите **F2**.

### ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №9

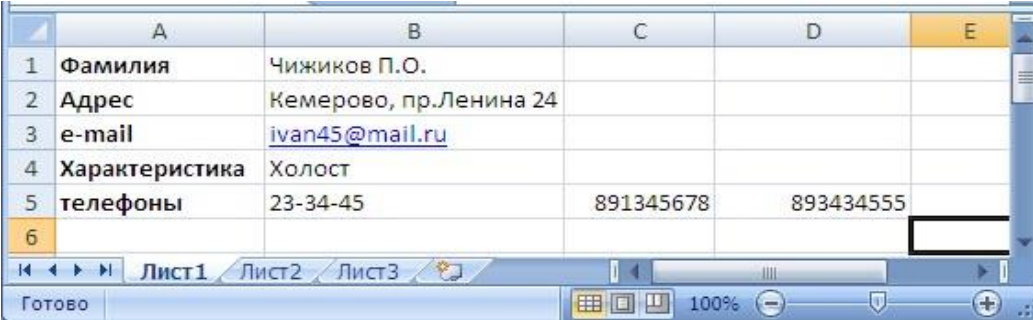
1. Найти сумму всех положительных чисел, записанных на Листе1.
2. На листе имеется область, ограниченная рамкой. Необходимо залить эту область определенным цветом.
3. Написать программу, которая вводимые пользователем данные об **n** альбомах исполнителя выводит на Лист1 Excel (рис. 9.1).



	А	В	С	Д	Е
1	Киркоров Ф.Б.				
2	Выпустил альбомы:				
3	«Дуэты»	12.12.2004	12000		
4	«For You»	01.03.2007	34444		
5	«ДруGOY»	10.05.2011	50000		
6	«Я»	11.09.2016	10000		
7					

Рисунок 9.1 – Лист Excel

4. Сформировать файл Excel для 5 сотрудников (рис. 9.2). Данные по каждому сотруднику разместить на отдельных листах Лист1...Лист15. Написать программу на VBA для считывания данных из файла Excel и вывода ее на экран.



The image shows a screenshot of an Excel spreadsheet with a blue-themed interface. The spreadsheet has five columns labeled A, B, C, D, and E. The first five rows contain data for a form. Row 6 is empty and highlighted. The bottom of the window shows the 'Лист1' tab, a status bar with 'Готово', and a zoom level of 100%.

	A	B	C	D	E
1	Фамилия	Чижиков П.О.			
2	Адрес	Кемерово, пр.Ленина 24			
3	e-mail	<a href="mailto:ivan45@mail.ru">ivan45@mail.ru</a>			
4	Характеристика	Холост			
5	телефоны	23-34-45	891345678	893434555	
6					

Рисунок 9.2 – Лист Excel



## **ПРАКТИЧЕСКАЯ РАБОТА №10. АВТОМАТИЗАЦИЯ СТАНДАРТНЫХ ТЕКСТОВЫХ ДОКУМЕНТОВ**

**Цель работы:** изучение основ работы со стандартными текстовыми документами.

### **ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ**

Ключевым в объектной модели Word является объект Application, так как он содержит все остальные объекты Word. Его элементами на разных уровнях иерархии являются около 180 объектов. Сам корневой объект Application имеет более сотни элементов: свойств, методов и событий.

#### **Свойства объекта Word.Application**

Свойства любого объекта делятся на две группы: свойства-участники (объекты) и терминальные свойства (обычные переменные VBA).

Единую систему организации панелей меню и инструментальных кнопок обеспечивает объект CommandBars, справочник – Assistant, поиска – FileSearch.

Центральными объектами Word являются коллекции Documents и Templates, точнее составляющие их элементы, сам документ и шаблоны. Объект AutoCorrect поддерживает работу по автоматической коррекции набираемых текстов. Его возможности эквивалентны команде Автозамена меню Сервис.

Объект Browser позволяет перемещать точку вставки, указывающую на объекты в документе. Коллекция объектов Dialogs представляет совокупность диалоговых окон, встроенных в Word. Добавлять новые или удалять элементы этой коллекции программным путем нельзя. Но соответствующие окна можно открыть и показать на экране дисплея и тем самым организовать диалог пользователем по теме, заданной соответствующим окном.

Три объекта, связанные с проверкой грамматики и орфографии: Languages, Dictionaries, SpellingSuggestions, – позволяют установить нужный язык, выбрать словарь, в том числе пользовательские словари, а также работать со списком слов, предлагаемых для исправления при обнаружении ошибки правописания. Команды Правописание и Язык меню Сервис

предоставляют аналогичные, функциональные возможности при работе с документом вручную.

С помощью объекта Options можно программным путем установить различные опции приложения и документа аналогично тому, как если бы вы выбрали команду Параметры в меню Сервис.

### **Работа с документами и класс Document**

Когда открывается приложение, создается коллекция документов Documents, содержащая все открытые документы. В начальный момент коллекция содержит минимум один новый или ранее существовавший документ. Новый документ добавляется методом Add, а уже существующий – методом Open объекта Documents. Чтобы добраться до нужного документа, достаточно указать его индекс – имя файла, хранящего документ, или его порядковый номер в коллекции. Для той же цели можно использовать и метод Item, но обычно он опускается. Метод Save позволяет сохранить документ, а метод Close, сохраняя документ в файле, закрывает его и удаляет из коллекции.

Глобальное свойство Dialogs возвращает коллекцию диалоговых окон. Константа wdDialogFileOpen задает конкретное диалоговое окно – объект класса Dialog.

### **Классы, задающие структуризацию текста документа**

Текст – это основа большинства документов. Его можно структурировать, оперируя различными единицами при решении тех или иных задач преобразования. Минимальной единицей текста обычно является символ. Кроме этого, существуют следующие единицы: слова, предложения, абзацы, а также более крупные образования: страницы, параграфы, главы.

Классы Characters, Words, Statements, Paragraphs, Sections позволяют работать с последовательностями (коллекциями) символов, слов, предложений, абзацев и разделов. Самой крупной единицей после абзаца выступает раздел. Элементом коллекций Characters, Words и Statements является объект класса Range. Объект Range позволяет работать как с одним элементом, так и с произвольной последовательностью элементов. Документы, поддокументы, абзацы, разделы – все они имеют

метод или свойство Range, возвращающее интервал, связанный с объектом. Поэтому работа с текстом так или иначе ведется через методы и свойства объекта Range.

### **Объект Document**

Объект Document может реагировать на три события, возникающие в результате действий пользователя. События объекта Document: Open, New, Close.

Рассмотрим основные классы, определяющие структуру документа.

**Subdocuments (Subdocument)** – коллекция и сам поддокумент. Есть некоторый разумный предел размера одного документа. Если в документе больше 10–20 страниц, работать с ним становится неудобно. В этом случае в нем выделяют главный документ и поддокументы. Главный документ в этом случае имеет коллекцию поддокументов, каждый из них является, по сути, документом, с которым можно работать независимо. Метод AddFromRange класса SubDocuments создает поддокумент, выделяя из главного документа область, заданную параметром Range.

**Tables (Table)**, **TablesOfAuthoritiesCategories (T.O.A.C)**, **TablesOfAuthorities (TableOfAuthorities)**, **TablesOfContents (TablesOfContent)**, **TablesOfFigures (TablesOfFigure)**. Класс Table определяет «обычные» таблицы с произвольным количеством строк и столбцов и произвольным заполнением полей. Остальные классы задают таблицы специального вида.

**Shapes(Shape)**, **InlineShapes(InlineShape)** – эти две коллекции с их элементами позволяют добавлять в документ рисунки, но не только их. ActiveX– и OLE-объекты также являются элементами этих коллекций. Элементы этих двух коллекций отличаются тем, как они привязаны к документу: первые могут свободно перемещаться, вторые жестко привязаны к заданной области документа.

**Lists(List)**, **ListParagraphs(ListParagraph)**, **listTemplates (ListTemplate)** – списки удобно вводить в документ, когда имеешь дело с перечислением. Списки можно оформлять в соответствии с шаблоном. Существуют две группы шаблонов: нумерованные списки и списки-бюллетени. Коллекция

ListTemplates содержит шаблоны оформления списков, а класс ListTemplate описывает конкретный шаблон. Шаблон применяется к списку абзацев и придает ему структуру, заданную шаблоном. Коллекция Lists содержит те списки документа (списки абзацев), что оформлены как нумерованные списки или списки-бюллетени. Коллекция ListParagraphs представляет список абзацев всех списков документа. Свойством ListParagraphs, которое возвращает объект соответствующего класса, обладает не только документ, но и объекты List и Range. Так что при наличии списка – объекта List можно выделить список абзацев. Чаще приходится выполнять обратную операцию – применять к списку абзацев один из возможных шаблонов, придав ему «настоящую» структуру списка. Тогда используют объект ListFormat.

**Comments(Comment)**, **Bookmarks(Bookmark)**, **FootNotes(FootNote)**, **EndNotes(EndNote)**, **Fields(Field)** – эти коллекции и их элементы отражают независимые, но близкие по духу понятия. Это части документа, косвенно связанные с ним. При нормальном просмотре документа они могут быть и не видны.

Коллекция comments и класс comment задают комментарии. Комментарии, как известно, вводятся для пояснения тех или иных терминов или понятий документа. Формально они приписываются некоторой области – объекту range.

Большой документ, к отдельным частям которого приходится часто обращаться, стоит снабдить закладками. Коллекция bookmarks задает все закладки данного документа.

Еще один способ комментирования – сноски. Они могут быть двух видов: подстраничные (внизу страницы) и концевые (в конце документа). Первые собраны в коллекцию footnotes, вторые – endnotes.

**Fields (Field)** – эта коллекция позволяет работать с полями документа. Одна из особенностей полей состоит в том, что их значения обновляются автоматически в зависимости от изменившихся внешних условий или контекста.

**Story Ranges (Range)** – эта коллекция представляет совокупность частей документа, называемых фрагментами (Story). Количество различных фрагментов документа

фиксировано. Нельзя добавлять элементы в эту коллекцию обычным способом, используя метод `Add`. Фрагменты появляются в коллекции, когда создается соответствующая часть документа. Фрагменты имеют тип, задаваемый константами из перечисления `wdStoryType`. Главный фрагмент – текст документа, тип которого задается константой `wdMainTextStory`. Комментарии, ссылки, колонтитулы составляют фрагменты других типов, т. е. сам фрагмент является объектом `Range`. Так что благодаря фрагментам можно, например, работать с коллекцией комментариев как с единой областью.

**Variables (Variable)** – с документом можно связать коллекцию переменных типа `Variant`. Это важная для программистов коллекция, так как время жизни переменных, в нее входящих, совпадает со временем жизни документа. Тем самым появляется возможность сохранять информацию о работе той или иной процедуры между сеансами. Например, можно иметь счетчики, подсчитывающие число вызовов макроса, и в зависимости от этого по-разному определять его дальнейшую работу.

### Объекты `Range` и `Selection`

Объект `Document` имеет метод `Range`, возвращающий объект `Range`, и метод `Select`, создающий объект `Selection`. Метод `Range` – это функция, возвращающая как результат объект `Range`; метод `Select` – это процедура без параметров, которая создает объект `Selection` в качестве побочного эффекта. Объект `Range` имеет метод `Select`, превращающий область объекта `Range` в выделенную. Тем самым метод `Select` определяет новый объект `Selection`. Симметрично, объект `Selection` имеет свойство `Range`, возвращающее объект `Range`, соответствующий выделенной области.

Большинство ранее описанных частей документа являются и частями (свойствами) объектов `Range` и `Selection`. Объект `Range` напоминает матрешку: в каждую область вложена область поменьше. Вот пример корректного (хоть и не самого эффективного) задания объекта `Range`: `ActiveDocument.Range.Sections(1).Range.Paragraphs(1).Range.Sentences(1)`.

Words(1).Characters(1)

### **Работа с текстом**

Объекты Range и Selection позволяют выполнять основные операции над текстом: «выделить», «добавить», «заменить», «удалить». У наших объектов большой набор методов, позволяющих реализовать эти операции. Все рассматриваемые здесь методы принадлежат обоим объектам, если не сделана специальная оговорка.

### **Выделение**

Выделить некоторую часть текста по существу означает определить объект Range или Selection. Объекты задают некоторую область в тексте документа, а их свойства Start и End позволяют установить начало и конец этой области. Меняя значения свойства, можно задать нужную область выделения.

Move является основным методом перемещения точки вставки. Остальные методы этой группы – в той или иной степени его модификации. Метод Move (Unit, Count) сжимает область в точку, стягивая ее в начало или конец, а затем перемещает точку вставки. Параметр Unit определяет единицы перемещения, а Count – количество этих единиц (по умолчанию 1). Знак переменной Count задает направление стягивания и перемещения. Положительные значения этого параметра задают стягивание к концу и перемещение вперед, отрицательные – стягивание в начало и перемещение назад. Чистое стягивание без перемещения точки вставки задается как перемещение на одну единицу. Метод возвращает количество единиц, на которое фактически произошло перемещение, или 0, если оно не осуществлено. Параметр Unit принимает значения wdCharacter (по умолчанию), wdWord, wdSentence, wdParagraph, wdSection, wdStory, wdCell, wdColumn, wdRow и wdTable.

Методы перемещения на сам текст не влияют – лишь изменяют область, заданную объектами Range и Selection. Поэтому эти методы применимы только к переменным типа Range, но не к фиксированным областям. Например, запись ActiveDocument.Paragraphs(1).Range.Move не имеет эффекта, поскольку область первого абзаца – вещь неизменяемая. Метод Move стягивает область в точку, которая и перемещается,

поэтому после его выполнения область исчезает, остается только точка вставки. Методы `MoveStart` и `MoveEnd` перемещают начальную или конечную точку области, обычно тем самым расширяя область.

### **Удаление текста**

Метод `Delete` позволяет удалить текст. Вызванный без параметров, он удаляет вызывающий его объект `Range` или `Selection`. Если он применен в форме `Delete (Unit, Count)`, удаляется часть текста в указанной области.

Параметр `Unit` задает единицы, но при удалении возможны только два значения: `wdWord` и `wdCharacter`. Параметр `Count` задает количество удаляемых единиц. Если область стянута в точку, удаляются символы перед точкой вставки или после нее в зависимости от знака параметра `Count`.

### **Вставка текста**

Группа методов `Insert` объектов `Range` и `Selection` позволяет осуществлять вставки в документ. Для вставки текста используются методы `InsertBefore(Text)` и `InsertAfter(Text)`. Параметр `text` типа `string` задает текст, вставляемый до или после области, заданной объектами `range` или `selection`.

После вставки текста область автоматически расширяется, включая в себя добавляемый текст.

Свойство `Text` позволяет заменять текст в выделенной области, поэтому нет нужды вызывать метод `Insert (Text)`. Методы `InsertBefore` и `InsertAfter` безопасны, так как текст добавляется, не изменяя содержимого области. Совсем иное дело – методы вставки, которые далеко не безопасны. При вставке внутрь области, например при использовании метода `InsertSymbol` или `InsertParagraph`, заменяется содержимое области.

### **Работа с буфером**

Метод `Copy`, не имеющий параметров, копирует объект (содержимое области) в буфер. Метод `cut`, действуя аналогично, должен бы заодно и удалять объект. Но сам объект не удаляется – только стягивается в точку, так что над ним возможны дальнейшие операции.

Иногда в буфер копируют не текст, а его формат. Этим занимается метод CopyFormat, копирующий формат по первому символу объекта selection. Если этот символ – метка абзаца, копируется формат абзаца. Методом CopyFormat обладает только объект selection.

Метод Paste позволяет поместить («приклеить») содержимое буфера в область, заданную объектами Range и Selection. Эта операция опасна, так как происходит замена, а не добавление текста. Поэтому обычно метод Paste применяется к объектам Range и Selection, предварительно стянутым в точку вставки. Метод PasteFormat применяет форматирование, хранящееся в буфере, к объекту Selection.

Наиболее важной особенностью работы на VBA в Word является вставка текста в документ при работе с приложениями. Для этого служат объекты Range и Selection, которые являются главными для практически любых операций, которые можно выполнять с помощью Word VBA. Некоторые из этих действий можно применять к документам в целом, но в общем случае вам необходим диапазон или выделенная область, прежде чем вносить изменения. Мы, однако, рассмотрим действия с документом при его создании.

Открытый документ Word уже содержит объекты Range, соответствующие многим его элементам. Каждый абзац, таблица, ячейка таблицы, комментарий и т. д. определяют диапазоны. Например, для того чтобы вставить некоторый текст в уже существующий документ, необходимо прописать код:

```
ActiveDocument.Paragraphs(1).Range.Text =
«Здравствуй, мир!»
```

Причем данная строка будет расположена в конце существующего параграфа. С другой стороны, используя объект Selection, можно также вставить некоторый текст в документ, используя метод Add и присвоение свойства Text объекту Selection:

```
If Documents.Count = 0 Then Documents.Add
Selection.Text = «Изучение работы с текстом в документе
Word является важной составной частью умения
```



программировать в VBA, « + TextBox1.Text +», и отвечает запросам всех программистов!»

Примечание:

**определение цвета**

Selection.Font.Color = wdColorRed – красный

wdColorDarkRed – бордовый

wdColorDarkTeal – бирюзовый

wdColorBlue – синий

wdColorGreen – зеленый

wdColorBlack – черный

wdColorOrange – оранжевый

**определение жирности**

Selection.Font.Bold = wdToggle – жирность

**определение начертания**

Selection.Font.Italic = wdToggle – курсив

**определение выравнивания**

Selection.ParagraphFormat.Alignment =

wdAlignParagraphRight – выравнивание по правому краю

wdAlignParagraphCenter – выравнивание по центру

wdAlignParagraphJustify – выравнивание по левому краю

**вставка в текст конкретного предложения**

Selection.TypeText Text:="Пример работы с текстом"

**вставка новой пустой строки**

Selection.TypeParagraph

**установка размера букв**

Selection.Font.Size = 14

## **ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №10**

1. Написать программу, которая позволяет найти количество слов "WORD" в текстовом документе. Изменить начертание, жирность и размер шрифта всех найденных слов на заданное пользователем.
2. Написать программу, которая позволяет вывести любой текст в текстовый документ в режиме "печатной машинки".
3. На лист табличного процессора поместить список студентов (ФИО, адрес, дата рождения). Всем военнообязанным студентам автоматически сформировать

письма, написав соответствующую программу на VBA (см. рис. 10.1). Дата явки определяется автоматически по формуле: Текущая дата+15дней.

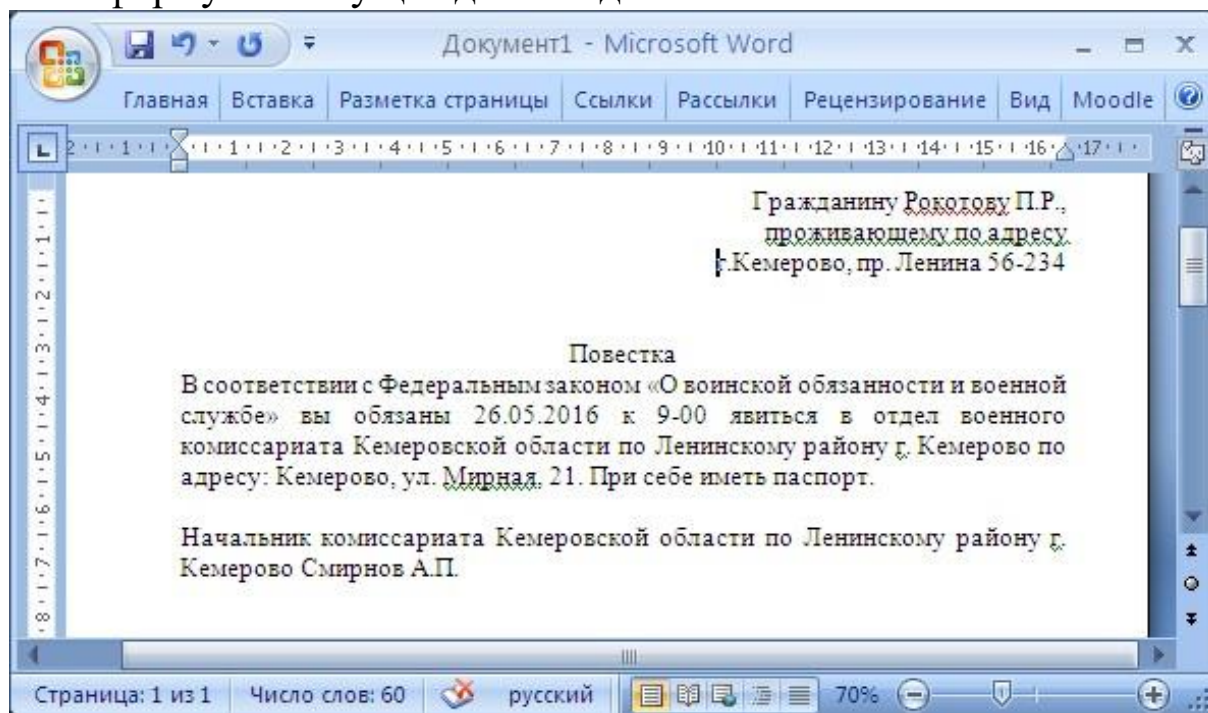


Рисунок 10.1 – Лист Excel

## ПРАКТИЧЕСКАЯ РАБОТА №11. ОРГАНИЗАЦИЯ ДОСТУПА К ОБЪЕКТАМ БАЗЫ ДАННЫХ

**Цель работы:** изучение способов обращения к основным объектам баз данных.

### ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

#### Программирование с использованием интерфейса DAO

Объекты доступа к данным (DAO, Data Access Object) – это иерархия объектов, обеспечивающая доступ к структуре базы данных и ее содержимому. В программах Visual Basic пользователь имеет возможность использовать объектный интерфейс DAO для выполнения следующих задач:

- осуществления доступа к данным в локальных и удаленных базах данных Access и внешних источниках;
- управления базой данных и ее объектами;
- изменения структуры таблиц и схемы данных;
- управления системой защиты;
- создания, настройки и синхронизации реплик.

Обе иерархии объектов DAO: для рабочей области Jet и для рабочей области **ODBCDirect** – начинаются с объекта DBEngine. Этот объект содержит свойства и методы, позволяющие управлять рабочими областями. Свойство **DefaultType** объекта DBEngine позволяет определять или устанавливать тип рабочей области, создаваемой по умолчанию.

Объект DBEngine в качестве свойства содержит семейство Workspaces всех открытых рабочих областей. Можно выбрать элемент этого семейства, указав индекс или имя, чтобы получить доступ к конкретной рабочей области.

Чтобы создать новую рабочую область ядра Jet или рабочую область ODBCDirect, используют метод CreateWorkspace объекта DBEngine с соответствующим параметром.

Несколько рабочих областей разных типов могут быть открыты одновременно. В момент создания рабочей области начинается сеанс работы с ней. После выполнения всех необходимых действий в рабочей области нужно завершить сеанс, закрыв рабочую область с помощью метода Close объекта Workspace.

Чтобы открыть базу данных, используйте существующий объект **Database** или создайте новый. Объект **Database** представляет собой базу данных Jet (файл MDB), базу данных ISAM или источник данных ODBC, подключенный через Jet.

Доступ к текущей базе данных осуществляется с помощью объекта типа **Database**, возвращаемого методом **CurrentDb** объекта **Application** (который представляет приложение Access). Метод **CurrentDb** входит в набор глобальных методов, поэтому для его вызова можно использовать сокращенную ссылку без префикса **Application** с точкой. Открыть существующую базу данных можно двумя способами:

- с помощью метода **OpenDatabase** объекта **Workspace**. В этом случае база данных будет открыта в заданной рабочей области;
- с помощью метода **OpenDatabase** объекта **DBEngine**. В этом случае база данных будет открыта в рабочей области, используемой по умолчанию.

Метод **OpenDatabase** объекта **DBEngine** входит в набор глобальных методов, поэтому при использовании сокращенной ссылки на этот метод без явного указания объекта (**DBEngine** или **Workspace**) используется метод объекта **DBEngine**. Метод **OpenDatabase** возвращает ссылку на созданный объект **Database** и имеет следующие параметры: **Database OpenDatabase(<имя>, <параметры>, <режим>, <соединение>)**

Метод **CreateDatabase** создает новый объект **Database**, добавляет его в семейство **Databases** открытых баз данных в рабочей области, сохраняет базу данных на диске и возвращает открытый объект **Database**. Этот метод используется только в рабочей области ядра Microsoft Jet. Метод **CreateDatabase** имеет следующие параметры:

**Database CreateDatabase (<имя>, <порядок>, <параметры>)**

Чтобы получить доступ к данным в открытой одним из перечисленных способов базе данных, необходимо открыть набор записей. Набор записей может представлять собой все записи таблицы или часть записей таблицы, удовлетворяющих указанному условию, или результат выборки из нескольких таблиц. Чтобы открыть набор записей в базе данных, используйте

метод **OpenRecordset** объекта Database. Этот метод возвращает ссылку на созданный объект Recordset и имеет следующие параметры (табл. 11.1):

Таблица 11.1 – Параметры метода OpenRecordset

Параметр	Тип	Обязательный или нет	Описание
<источник>	String	Обязательный	Источник записей для нового объекта Recordset. В качестве источника записей можно указать имя таблицы или запроса, а также инструкцию SQL, которая возвращает записи
<тип>	<константа>	Необязательный	Константа, указывающая тип открываемого объекта Recordset.
<параметры>	<константы>	Необязательный	Произвольная комбинация определенных констант, задающих характеристики нового объекта Recordset. Эти константы приведены в справочной системе Access
<блокировки>	<константа>	Необязательный	Константа, определяющая тип блокировки объекта Recordset. Возможные константы перечислены в справке Access

С помощью объектов DAO можно осуществлять всевозможные манипуляции с данными. Например, такие как:

- получение набора записей из таблицы;
- выполнение запроса SQL для получения набора записей;
- использование запроса, сохраненного в базе данных в виде объекта;
- использование набора методов объекта Recordset, предоставляющего широкие возможности по обработке данных: чтение, анализ и изменение данных без составления инструкций на языке SQL.

В следующих примерах приведены приемы программного изменения данных в открытом наборе записей. Переменная rs соответствует открытому набору записей – объекту типа Recordset.

**Пример. Добавление записи в таблицу**

'Добавляем сообщение в таблицу сообщений клиента

rs.AddNew 'Создание новой записи

rs!ИмяИгрока = playerName 'Запись значения в поле  
ИмяИгрока rs!Сообщение = message 'Запись значения в поле  
Сообщение

rs.Update 'Сохранение изменений в источнике

rs.Bookmark = rs.LastModified

'Перемещение курсора на новую запись

**Пример. Изменение текущей записи в таблице**

'Увеличиваем счет игрока, сделавшего ход

'Делаем текущей запись, содержащую данные для  
нужного игрока rs.FindFirst "[ИмяИгрока] = '" &  
newPlayer & ""

rs.Edit 'Переводим запись в режим правки

rs!Счет = rs!Счет + newCount 'Изменяем значение  
поля Счет rs.Update 'Сохраняем изменения

rs.Bookmark = rs.LastModified 'Делаем измененную запись  
текущей

**Пример. Удаление текущей записи в таблице**

playerQueryCode = rs!КодЗаявки 'Сохраняем параметры  
заявки

playerTrial = rs!Значение 'во временных переменных

rs.Delete 'и удаляем заявку из таблицы

Для перемещения по записям используются методы **Move**, **MoveFirst**, **MoveNext**, **MovePrev**, **MoveLast** объекта Recordset. Метод MoveLast перемещает курсор на последнюю запись в наборе, что приводит к загрузке в набор всех записей. После этого можно прочитать значение свойства **Count** объекта Recordset. Оно будет соответствовать общему количеству записей в наборе.

### **ЗАДАНИЕ К ПРАКТИЧЕСКОЙ РАБОТЕ №11**

Написать программу на языке C#. Вся информация по сотрудникам находится в базе данных.

1. Вывести на Лист1 табличного процессора телефоны всех сотрудников.
2. Вывести на Лист2 табличного процессора всех сотрудников отдела, название которого введено с клавиатуры.
3. Найти среднюю зарплату сотрудников экономического отдела. Результат вывести на экран в диалоговое окно.
4. Найти номера телефонов и их количество у всех сотрудников банка. Результат вывести в текстовый файл.
5. Вывести количество сотрудников экономического отдела, имеющие должность «инженер». Результат вывести на экран в диалоговое окно.
6. Вывести количество оценок 3, 4, 5 по ТБ у сотрудников финансового отдела. Результат вывести на экран в диалоговое окно.

## ПРАКТИЧЕСКАЯ РАБОТА №12. РАЗРАБОТКА ПРИЛОЖЕНИЯ ДЛЯ РЕШЕНИЯ ПРИКЛАДНЫХ ЗАДАЧ

**Цель работы:** изучение основ разработки прикладных программ. Сформировать умения использования контейнерных элементов управления при создании проекта VisualStudio, изучить их основные свойства; сформировать умения по созданию процедур на основе событий компонентов.

### ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

#### Панель вкладок TabControl в С# и платформе .NET

Создаем программу - тест.

Чтобы настроить вкладки элемента TabControl используем свойство TabPages. При переносе элемента TabControl с панели инструментов на форму по умолчанию создаются две вкладки - tabPage1 и tabPage2 (рис. 12.1-12.4).

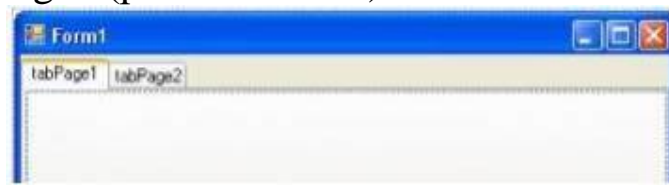


Рисунок 12.1 – Вкладки TabControl

Изменим их отображение с помощью свойства TabPages:



Рисунок 12.2 – Изменение отображения

Нам откроется окно редактирования/добавления и удаления вкладок:

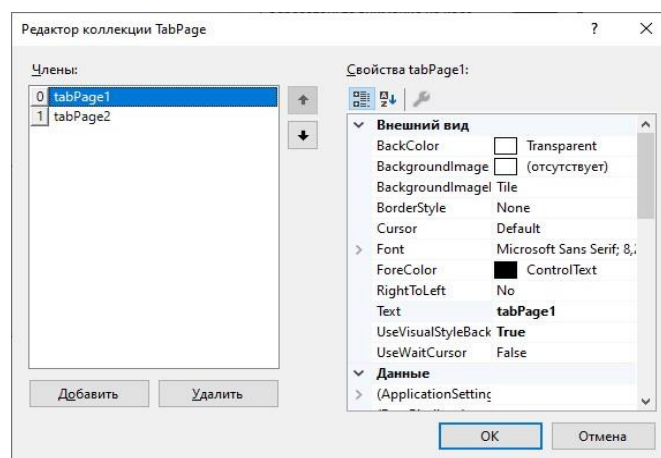


Рисунок 12.3 – Редактор коллекции



Каждая вкладка представляет своего рода панель, на которую мы можем добавить другие элементы управления, а также заголовок, с помощью которого мы можем переключаться по вкладкам. Текст заголовка задается с помощью свойства Text.

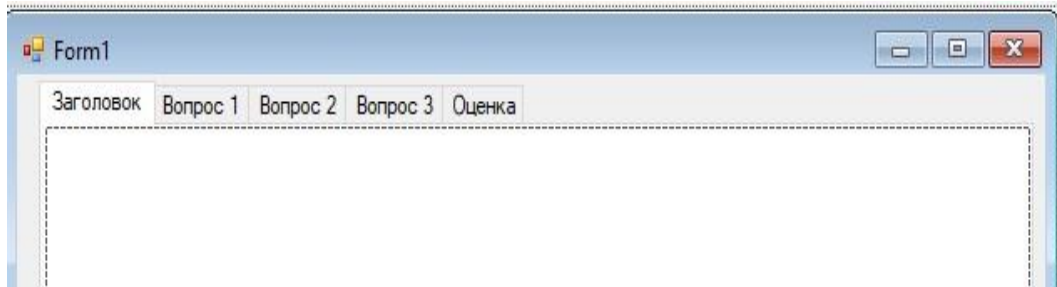


Рисунок 12.4 – Вкладки TabControl

### 1. Ответ вводится в виде числа, слова.

На основе этих вкладок создадим простейшую программу тест (рис.12.5).

```

Ссылка: 3
public partial class Form1 : Form
{
    // Опишем глобальную переменную для хранения количества правильных ответов b
    int b = 0;

    Ссылка: 1
    public Form1()
    {
        InitializeComponent();
        // Добавим команду, вывода формы в центре экрана.
        this.StartPosition = FormStartPosition.CenterScreen;
    }

    Ссылка: 0
    private void button1_Click(object sender, EventArgs e)
    {
        if (textBox1.Text == "4")
        {
            label1.Text = "Правильно";
            b = b + 1;
        }
        else label1.Text = "Неправильно";
        // прячем кнопку что бы исключить повторный ввод
        // (угадывание ответа)
        button1.Visible = false;
    }
}

```

Рисунок 12.5 – Код программы

Разместим на вкладке следующие элементы. Номера элементов могут быть другие.

Все зависит от количества ранее установленных элементов.

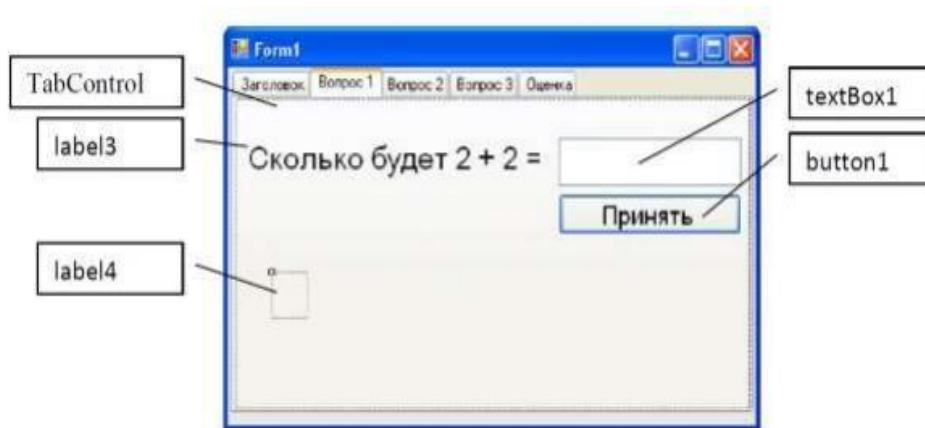


Рисунок 12.6 – Вкладки TabControl

Процедура «Принять» будет иметь следующий вид (см. функцию button1\_Click).

## 2. Выбирается один вопрос при помощи радиокнопок.

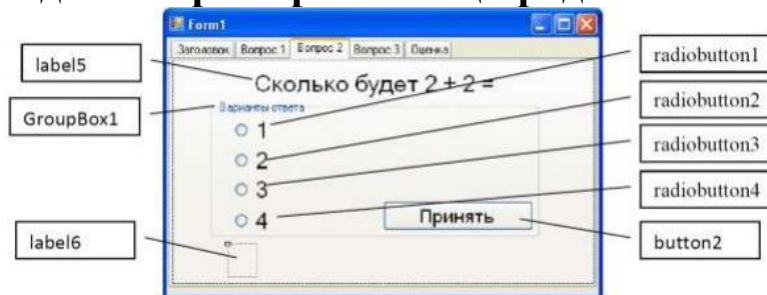


Рисунок 12.7 – Вкладки TabControl

Процедура «Принять» будет иметь следующий вид:

```
private void button2_Click(object sender, EventArgs e)
{
    if (radioButton4.Checked == true)
    {
        label6.Text = "Правильно";
        b = b + 1;
    }
    else label6.Text = "Неправильно";
    button2.Visible = false;
}
```

Рисунок 12.8 – Процедура «Принять»

## 3. Выбираем несколько ответов.

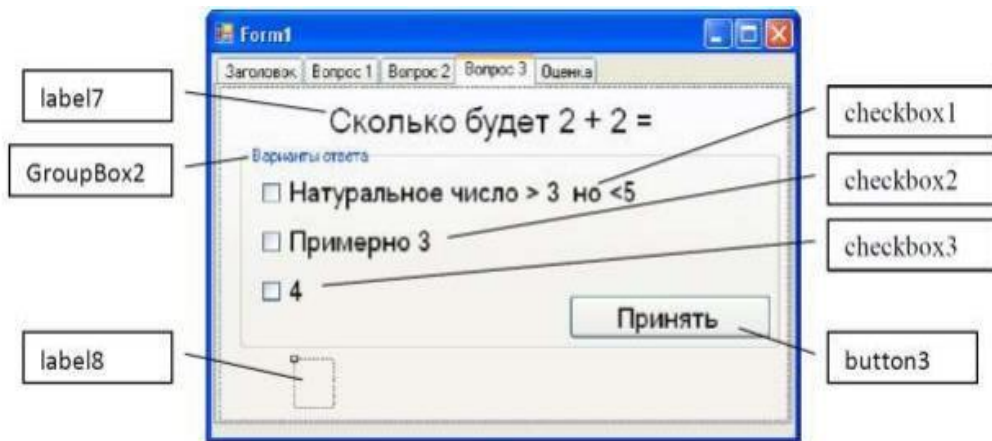


Рисунок 12.9 – Вкладки TabControl

Процедура «Принять» будет иметь следующий вид:

```
private void button3_Click(object sender, EventArgs e)
{
    if (checkBox1.Checked == true && checkBox3.Checked == true && checkBox2.Checked == false)
    {
        label8.Text = "Правильно"; b = b + 1;
    }
    else label8.Text = "Неправильно";
    button3.Visible = false;
}
```

Рисунок 12.10 – Процедура «Принять»

Так как галочки можно поставит на всех checkbox, то проверяем их все, не поставлено ли лишних галочек. На вкладке «Оценка» предлагаем три возможных варианта:



Рисунок 12.11 – Вкладки TabControl

Процедура «Получить» оценку может выглядеть так:

```

label9.Text = "Набрано баллов= " + Convert.ToString(b);
if (b == 10)
    label10.Text = "Оценка 5 (отлично)";
if (b == 9 || b == 8 || b == 7)
    label10.Text = "Оценка 4(хорошо)";
if (b == 6 || b == 5)
    label10.Text = "Оценка 3(удовлетворительно)";
if (b == 4 || b == 3)
    label10.Text = "Оценка 2(плохо)";
if (b == 2 || b == 1)
    label10.Text = "Оценка 1(все ужасно)";

```

Рисунок 12.12 – Процедура «Получить»

Переход в начало потребует очистки всех введенных пользователем данных:

```

// переходим на первую вкладку
tabcontrol1.SelectedIndex = 0;
// показываем все кнопки «Проверить» обратно. button1.Visible = true;
button2.Visible = true;
...
button10.Visible = true;
// скрываем надписи правильно неправильно
label20.text = "";
...
label2.text = "";
// убираем все галочки кнопочки и надписи в полях ответа checkBox1.checked= false;
...
checkBox12.checked:= false; textBox1.Text = "";
...
textBox3.Text = "";
radioButton1.checked= false;
...
radioButton14.checked= false;
// обнуляем балы и изменяем соответствующие надписи b = 0;
label21.Text = "Набрано баллов= ";
label22.Text = "";

```

В вопросе можно использовать картинки. Пример оформления вопроса:



Рисунок 12.13 – Вкладки TabControl

### ЗАДАНИЕ ДЛЯ ПРАКТИЧЕСКОЙ РАБОТЫ №12

Создать приложение для решения прикладных задач:

1. Программа выводит текущее время и текущую дату.
2. Программа «Электронные часы», в окне которой отображается текущее время, дата и день недели, календарь.
3. Программу «Светофор», которая через определенные интервалы меняет свой цвет.

## САМОСТОЯТЕЛЬНАЯ РАБОТА №1

### Раздел 1. Основы алгоритмизации

#### Тема 1.1. Алгоритмизация вычислительного процесса

Содержание темы:

1. Понятие алгоритма.
2. Способы описания алгоритмов.
3. Линейный разветвляющийся, циклический алгоритм.

### МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

В процессе изучения данного раздела студент должен усвоить, что **программное обеспечение (ПО)** – это совокупность программных средств для обеспечения нормальной работы вычислительной системы, подразделяющееся на общее и прикладное программное обеспечение. Все программное обеспечение можно разделить на три вида:

- системное ПО
- средства разработки
- прикладные программы

**Системное программное обеспечение** – это операционные системы, различные программы-утилиты для диагностики ресурсов компьютера (например, тестирования оперативной памяти), предоставления пользователю удобного способа работы взаимодействия с компьютером (например, командная строка), а также обслуживания ресурсов компьютера (например, разметка диска).

К **средствам программирования** относятся множество языков программирования, средства для автоматизации процесса создания программ, компиляторы и интерпретаторы. Языки и системы программирования являются по своему назначению инструментами для создания действительно полезного ПО. С их помощью создается как прикладное, так и системное программное обеспечение, а также новые средства разработки.

Огромную долю в ПО занимают **прикладные программы**, которые в свою очередь делят на **универсальные** и **специализированные**.

Перед созданием любого программного обеспечения необходимо построить алгоритм его функционирования. **Алгоритм** – это понятное и точное предписание совершить определенную последовательность действий, направленную на достижение указанной цели или решение поставленной задачи. **Алгоритм применительно к вычислительной машине** – точное предписание, т. е. набор операций и правил их чередования, при помощи которого, начиная с некоторых исходных данных, можно решить любую задачу фиксированного типа.

Любой алгоритм характеризуется следующими свойствами:

- **Дискретность** - алгоритм должен быть разбит на шаги (отдельные законченные действия).
- **Определенность** - у исполнителя не должно возникать двусмысленностей в понимании шагов алгоритма (исполнитель не должен принимать самостоятельные решения).
- **Результативность (конечность)** - алгоритм должен приводить к конечному результату за конечное число шагов.
- **Понятность** - алгоритм должен быть понятен для исполнителя.
- **Эффективность** - из возможных алгоритмов выбирается тот алгоритм, который содержит меньше шагов или на его выполнение требуется меньше времени.

В процессе алгоритмизации исходный алгоритм разбивается на отдельные связанные части, называемые шагами, или **частными алгоритмами**. Различают четыре основных типа частных алгоритмов:

- линейный алгоритм;

- алгоритм с ветвлением;
- циклический алгоритм;
- вспомогательный, или подчиненный, алгоритм.

**Линейный алгоритм** – набор инструкций, выполняемых последовательно во времени друг за другом.

**Алгоритм с ветвлением** – алгоритм, содержащий хотя бы одно условие, в результате проверки которого ЭВМ обеспечивает переход на один из двух возможных шагов.

**Циклический алгоритм** – алгоритм, предусматривающий повторения одного и того же действия над новыми исходными

данными. Необходимо заметить, что циклический алгоритм легко реализуется посредством двух ранее рассмотренных типов алгоритмов.

**Вспомогательный, или подчиненный, алгоритм** – алгоритм, ранее разработанный и целиком используемый при алгоритмизации конкретной задачи.

В настоящее время различают несколько способов описания алгоритмов:

1. **Словесный**, то есть записи на естественном языке, описание словами последовательности выполнения алгоритма.

Например: записать алгоритм нахождения наибольшего общего делителя (НОД) двух натуральных чисел. Алгоритм может быть следующим: задать два числа; если числа равны, то взять любое из них в качестве ответа и остановиться, в противном случае продолжить выполнение алгоритма; определить большее из чисел; заменить большее из чисел разностью большего и меньшего из чисел; повторить алгоритм с шага.

2. **Формульно-словесный**, аналогично пункту 1, плюс параллельная демонстрация используемых формул.

В качестве примера можно привести ведение лекций преподавателем (словесный способ) с одновременной записью формул на доске (формульный).

3. **Графический**, то есть с помощью блок-схем (рис. 1).

Графический способ представления алгоритмов является более компактным и наглядным по сравнению со словесным. При графическом исполнении алгоритм изображается в виде последовательности связанных между собой блочных символов, каждый из которых соответствует выполнению одного из действий. Такое графическое представление называется схемой алгоритма или блок-схемой. В блок-схеме каждому типу действий (вводу исходных данных, вычислению значений выражений, проверке условий, управлению повторением действий, окончанию обработки и т.п.) соответствует геометрическая фигура, представленная в виде блочного символа. Блочные символы соединяются линиями переходов, определяющими очередность выполнения действий.



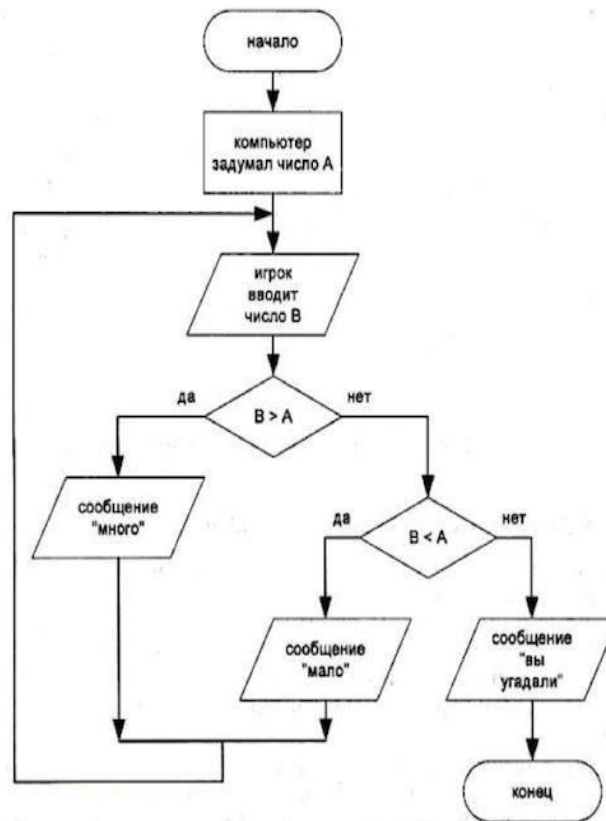


Рисунок 1.1 – Описание алгоритма с помощью блок-схемы

4. **Программный**, то есть тексты на языках программирования, предназначенные для исполнения на компьютере.

### КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Дайте определение алгоритму.
2. Свойства алгоритма.
3. Способы записи алгоритма.
4. Основные элементы блок-схемы.
5. Виды алгоритмов.
6. Отличительные особенности алгоритмов с предусловием и постусловием.
7. Какие цели преследуются при разработке прикладного программного обеспечения?
8. Зачем нужна алгоритмизация вычислительного процесса?
9. Какие способы описания алгоритмов вы знаете?
10. Какие блоки используются при графическом способе описания алгоритма?

## САМОСТОЯТЕЛЬНАЯ РАБОТА №2

### Раздел 2. Введение в программирование

#### Тема 2.1. Языки программирования

Содержание темы:

1. Развитие языков программирования.
2. Обзор языков программирования. Области применения языков программирования. Стандарты языков программирования. Среда проектирования.
3. Компиляторы и интерпретаторы.
4. Жизненный цикл программы. Программа. Программный продукт и его характеристики.
5. Основные этапы решения задач на компьютере.

#### Тема 2.2. Типы данных

Содержание темы:

1. Типы данных. Простые типы данных.
2. Производные типы данных.
3. Структурированные типы данных.

## МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

### Языки программирования

Существуют различные классификации языков программирования. По наиболее распространенной классификации все языки программирования, в соответствии с тем, в каких терминах необходимо описать задачу, делят на языки **низкого** и **высокого уровня**.

Если язык близок к естественному языку программирования, то он называется языком высокого уровня, если ближе к машинным командам, – языком низкого уровня.

В группу языков низкого уровня входят машинные языки и языки символического кодирования: Автокод, Ассемблер. Операторы этого языка – это те же машинные команды, но записанные мнемоническими кодами, а в качестве операндов используются не конкретные адреса, а символические имена. Все языки низкого уровня ориентированы на определенный тип компьютера, то есть являются машинно-зависимыми.

К языкам программирования высокого уровня относят Фортран (переводчик формул – был разработан в середине 50-х годов программистами фирмы IBM и в основном используется для программ, выполняющих естественно-научные и математические расчеты), Алгол, Кобол (коммерческий язык – используется, в первую очередь, для программирования экономических задач), Паскаль, Бейсик (был разработан профессорами Дармутского колледжа Джоном Кемени и Томасом Курцом.), Си, Пролог (в основе языка лежит аппарат математической логики) и т.д.

Эти языки машинно-независимы, так как они ориентированы не на систему команд той или иной ЭВМ, а на систему операндов, характерных для записи определенного класса алгоритмов. Однако программы, написанные на языках высокого уровня, занимают больше памяти и медленнее выполняются, чем программы на машинных языках.

Программу, написанную на языке программирования высокого уровня, ЭВМ не понимает, поскольку ей доступен только машинный язык. Поэтому для перевода программы с языка программирования на язык машинных кодов используют специальные программы-трансляторы.

Существует три вида транслятора: **интерпретаторы** (это транслятор, который производит пооператорную обработку и выполнение исходного кода программы), **компиляторы** (преобразует всю программу в модуль на машинном языке, после чего программа записывается в память компьютера и лишь потом исполняется) и **ассемблеры** (переводят программу, записанную на языке ассемблера, в программу на машинном языке).

Языки программирования также можно разделять на поколения: – языки первого поколения: машинно-ориентированные с ручным управлением памяти на компьютерах первого поколения.

– языки второго поколения: с мнемоническим представлением команд, так называемые автокоды.

– языки третьего поколения: общего назначения, используемые для создания прикладных программ любого типа. Например, Бейсик, Кобол, Си и Паскаль.

- языки четвертого поколения: усовершенствованные, разработанные для создания специальных прикладных программ, для управления базами данных.

- языки программирования пятого поколения: языки декларативные, объектно–ориентированные и визуальные. Например, Пролог, ЛИСП (используется для построения программ с использованием методов искусственного интеллекта), Си++, Visual Basic, Delphi.

Языки программирования также можно классифицировать на процедурные и непроцедурные.

**В процедурных языках** программа явно описывает действия, которые необходимо выполнить, а результат задается только способом получения его при помощи некоторой процедуры, которая представляет собой определенную последовательность действий. Среди процедурных языков выделяют в свою очередь структурные и операционные языки. В структурных языках одним оператором записываются целые алгоритмические структуры: ветвления, циклы и т.д. В операционных языках для этого используются несколько операций. Широко распространены следующие структурные языки: Паскаль, Си, Ада, ПЛ/1. Среди операционных известны Фортран, Бейсик, Фокал.

**Непроцедурное (декларативное)** программирование появилось в начале 70-х годов 20 века. К непроцедурному программированию относятся функциональные и логические языки.

В функциональных языках программа описывает вычисление некоторой функции. Обычно эта функция задается как композиция других, более простых, те в свою очередь делятся на еще более простые задачи и т.д. Один из основных элементов функциональных языков – рекурсия. Оператора присваивания и циклов в классических функциональных языках нет.

В логических языках программа вообще не описывает действий. Она задает данные и соотношения между ними. После этого системе можно задавать вопросы. Машина перебирает известные и заданные в программе данные и находит ответ на вопрос. Порядок перебора не описывается в программе, а неявно

задается самим языком. Классическим языком логического программирования считается Пролог. Программа на Прологе содержит, набор предикатов–утверждений, которые образуют проблемно– ориентированную базу данных и правила, имеющие вид условий.

Можно выделить еще один класс языков программирования – объектно–ориентированные языки высокого уровня. На таких языках не описывают подробной последовательности действий для решения задачи, хотя они содержат элементы процедурного программирования. Объектно– ориентированные языки, благодаря богатому пользовательскому интерфейсу, предлагают человеку решить задачу в удобной для него форме. Первый объектно-ориентированный язык программирования Simula был создан в 1960-х годах Нигаардом и Далом.

### **КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Опишите семантику языков программирования.
2. Чем отличаются компилируемые и интерпретируемые языки?
3. Перечислите первые языки программирования высокого уровня.
4. Перечислите и дайте краткое описание известных вам алгоритмических языков программирования.
5. Какие методы используются при создании программного кода?
6. Перечислите основные виды ПО и его назначение.
7. Перечислите основные этапы разработки прикладного программного обеспечения.
8. В чем заключается постановка задачи оптимизации?
9. Какие языки программирования вы знаете?

### **САМОСТОЯТЕЛЬНАЯ РАБОТА №3**

#### **Раздел 3. Объектно-ориентированное программирование**

##### **Тема 3.1. Основные принципы объектно-ориентированного программирования (ООП)**

Содержание темы:

1. История развития ООП. Базовые понятия ООП: объект, его свойства и методы, класс, интерфейс.
2. Основные принципы ООП: инкапсуляция, наследование, полиморфизм.
3. Классы объектов. Компоненты и их свойства.
4. Событийно-управляемая модель программирования. Компонентноориентированный подход.

##### **Тема 3.2. Интегрированная среда разработчика.**

Содержание темы:

1. Требования к аппаратным и программным средствам интегрированной среды разработчика.
2. Интерфейс среды разработчика: характеристика, основные окна, инструменты, объекты. Форма и размещение на ней управляющих элементов.
3. Панель компонентов и их свойства. Окно кода проекта.
4. Состав и характеристика проекта. Выполнение проекта. Настройка среды и параметров проекта.
5. Панель компонентов и их свойства. Окно кода проекта. Состав и характеристика проекта. Выполнение проекта. Настройка среды и параметров проекта.
6. Настройка среды и параметров проекта.

##### **Тема 3.3. Визуальное событийно-управляемое программирование**

Содержание темы:

1. Основные компоненты (элементы управления) интегрированной среды разработки, их состав и назначение.
2. Дополнительные элементы управления. Свойства компонентов. Виды свойств. Синтаксис определения

свойств. Назначения свойств и их влияние на результат. Управление объектом через свойства.

3. События компонентов (элементов управления), их сущность и назначение. Создание процедур на основе событий.

## **МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**

В процессе изучения данного раздела студент должен усвоить основы работы в среде Visual Studio. Для этого студенту необходимо самостоятельно изучить теоретический материал:

1. Интегрированная среда разработки приложений Visual Studio

- Изучение среды и основ работы в Visual Studio
- Запуск Visual Basic и создание нового проекта
- Основные компоненты интерфейса Visual Studio
- Текстовое меню и панель инструментов
- Окна: конструктора форм и панель элементов управления
- Создание формы приложения и сохранение проекта
- Запуск приложения и работа с окном редактора исходного кода

- Этапы создания простейшего приложения

2. Элементы управления (ЭУ)

- Виды элементов управления

В данном подразделе студенту необходимо обратить внимание на наиболее часто используемые ЭУ, приведённые в таблице 3.1.

Таблица 3.1 – Элементы управления

<b>Название элемента управления</b>	<b>Описание элемента управления</b>
Командная кнопка CommandButton	Используется для выполнения действий. К каждой кнопке присоединяется процедура, которая выполняется тогда, когда пользователь щелкает по кнопке
Ярлык (надпись) Label	Ярлык добавляет к форме любой текст

Текстовое окно TextBox	Служит для ввода пользователем любых данных
Кнопки–переключатели OptionButton	Обеспечивают пользователю возможность выбрать одну опцию из нескольких
Контрольные индикаторы CheckBox	Обеспечивают пользователю возможность выбрать любое количество опций
Рамка Frame	Позволяет объединять элементы управления в группы, которые будут работать независимо друг от друга
Линейки прокрутки	Используются для вертикальной или горизонтальной прокрутки данных в текстовых окнах
Окна списков ListBox	Позволяет пользователю выбирать из списка возможностей
Комбинированные списки ComboBox	Комбинация текстового окна и окна списка. Элемент может быть выбран из списка либо щелчком по нему, либо в печатывание имени в текстовое поле
Изображение Image	Выводит на экран содержимое графического файла
Данные Data	Осуществляет доступ к данным
Контур Shape	Создает на форме прямоугольник, квадрат, овал, окружность, прямоугольник и квадрат со скругленными углами
Рисунок Picture	Предназначен для отображения графических изображений; в качестве контейнера для других элементов управления; в виде графического окна для вывода текста, графических элементов
Линия Line	Добавляет на форму линию
Таймер Timer	Обрабатывает данные системных часов

- Свойства элементов управления.

В данном подразделе студенту необходимо обратить внимание на то, что свойства любого объекта можно установить во время проектирования в окне Properties, или во время



выполнения, написав соответствующий программный код:  
Объект.Свойство=значение.

Каждый из размещаемых в форме элементов управления определяется собственным набором свойств. Но есть свойства, присущие большинству объектов. К ним относятся такие свойства, как Name, Height, Width, Left, Top, Visible, Caption и т.д.

- **Методы**

В данном подразделе студент должен усвоить, что помимо свойств, объект имеет методы, определяющие выполняемые им действия. Вызов метода осуществляется по правилам:

- не требующего ввода аргумента: объект.метод
- требующего ввода аргумента: объект.метод аргумент1, аргумент2,..., аргумент n
- возвращающего значение:

переменная=объект.метод (аргумент1, аргумент2, ..., аргумент n). Каждый объект имеет некоторое количество доступных ему методов.

- **События**

В данном подразделе студент должен уяснить, что помимо свойств и методов, для объектов можно задать программные коды, написанные на языке Visual Basic и выполняемые при наступлении связанных с ними событий. Наиболее часто используемые события приведены в таблице 3.2.

Таблица 3.2 – События

Событие	Описание
Click (щелчок)	Происходит при щелчке пользователя кнопкой мыши по объекту
Double Click (двойной щелчок)	Происходит при двойном щелчке пользователя кнопкой мыши по объекту
Change	Происходит при изменении объекта. Например, при вводе текста.
Activate	Происходит при загрузке формы

## **КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Какие основные объекты интегрированной среды программирования Visual Studio вы знаете?
2. Какие стандартные объекты используются при создании прикладной программы?
3. Перечислите наиболее часто используемые свойства объектов.
4. Что такое «метод»?
5. Что такое «тело метода»?
6. Перечислите наиболее часто используемые события.
7. Какие типы данных существуют в C#?

## **САМОСТОЯТЕЛЬНАЯ РАБОТА №4**

### **Раздел 4. Операторы языка программирования**

#### **Тема 4.1. Операторы языка программирования**

Содержание темы:

1. Операции и выражения. Правила формирования и вычисления выражений. Ввод и вывод данных.
2. Оператор присваивания. Составной оператор.
3. Условный оператор. Оператор выбора.
4. Цикл с постусловием. Цикл с предусловием. Цикл с параметром. Вложенные циклы.
5. Строки. Стандартные процедуры и функции для работы со строками.
6. Массивы. Двухмерные массивы.

### **МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**

В процессе изучения данного раздела студент должен усвоить основные операторы языка программирования. Для этого студенту необходимо самостоятельно изучить теоретический материал

1. Работа с переменными
  - Объявление переменных
  - Присвоение значения переменным

В данном подразделе студент должен уяснить, что переменная – это поименованная ячейка памяти, хранящая какое-либо одно значение (одно число, один фрагмент текста). Имя переменной – это строка символов, которая отличает ее от других переменных и объектов программы (элементов управления). Значение переменной – это данные, которые в ней хранятся. Тип данных (тип переменной) обуславливает то, как хранятся и обрабатываются данные.

Кроме того, студент должен разобраться с синтаксисом операторов, используемых для объявления переменных: **PROTECTED**, **STATIC**, **PUBLIC**, **PRIVATE**; уяснить необходимость использования того или иного типа данных (**INT**, **LONG**, **DOUBLE**, **BOOLEAN**, **BYTE**, **STRING**, **VAR**).

## 2. Разработка приложений с разветвляющимися структурами алгоритмов

- Разветвляющиеся алгоритмы
- Программирование условий в C#
- Логические выражения

В данном подразделе студент должен научиться реализовывать разветвляющийся алгоритм для выполнения различных фрагментов программы в зависимости от выполнения некоторого условия. Разветвляющийся алгоритм реализуется с помощью условного оператора `if . . . else`, который может иметь простую однострочную или блочную структуру.

## 3. Разработка приложений с циклическими структурами алгоритмов

- Циклические алгоритмы
- Виды циклов
- Программирование циклов в C#

В данном разделе студент должен научиться работать с циклами `FOR` (циклы с заданным количеством повторений, которые позволяют выполнять группу операторов заданное число раз) и с циклами `While`, `Do...While`, которые предназначены для ситуаций, когда количество проходов цикла заранее не известно, но зато известно условие выхода из цикла.

## 4. Массивы

- Основные сведения о массивах
- Одномерные массивы
- Двумерные массивы
- Динамические массивы

В данном разделе студент должен научиться объявлять одномерные и многомерные массивы; заполнять массивы данными; выводить массивы на экран; выполнять с массивами различные операции (нахождение максимального элемента, суммы всех элементов, количества элементов и т.д.).

## 5. Строковые данные

- Функции обработки строк
- Работа со строковым типом данных.

В данном разделе студент должен научиться работать с функциями обработки строк.

### **КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Какие функции работы со строками вы знаете?
2. Как осуществляется сложение строк?
3. Что такое массив?
4. Какие массивы вы знаете?
5. Какие способы формирования (заполнения) массивов вы знаете?
6. Как обратиться к элементу массива?
7. Какие основные математические операторы Visual Basic вы знаете?
8. Какие основные функции, существующие в Visual Basic, вы знаете?
9. Чем отличаются циклы с заданным повторением от циклов по условию?
10. Какие виды условий существуют?
11. Какие операторы используются при написании условных выражений?
12. Какие операции можно совершать с условными выражениями?
13. Зачем используются массивы данных в организации дорожного движения?

## САМОСТОЯТЕЛЬНАЯ РАБОТА №5

### Раздел 5. Структурное и модульное программирование

#### Тема 5.1. Процедуры и функции

Содержание темы:

1. Общие сведения о подпрограммах. Определение и вызов подпрограмм.
2. Область видимости и время жизни переменной.
3. Механизм передачи параметров. Организация функций.
4. Рекурсия. Программирование рекурсивных алгоритмов.

#### Тема 5.2. Структурное и модульное программирование

Содержание темы:

1. Основы структурного программирования. Методы структурного программирования.
2. Модульное программирование. Понятие модуля. Структура модуля.
3. Компиляция и компоновка программы.
4. Стандартные модули.

## МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ

При изучении данного раздела студент должен уяснить, что **модульное программирование** – это независимое программирование каждого модуля. Программирование обычно ведут начиная с верхнего уровня. Вначале составляется программа для ограниченного числа модулей верхнего уровня и производится тестирование всей задачи, при этом остальные модули не программируются, а для них делают так называемые заглушки.

Каждый новый модуль включается в основную программу только после того, как проведено его полное тестирование.

**Структурное программирование (кодирование)** – процесс программирования на алгоритмическом языке с использованием определенных конструкций.

Основные положения структурного программирования:

1. Любая программа составляется на базе основных алгоритмических структур трех видов: линейного, разветвляющегося и циклического.

2. Между этими структурами производится передача управления только вперед, что соответствует в блок-схеме направлению линий сверху вниз.

3. Недопустимо пользоваться в программе специальной командой безусловной передачи управления из одного места программы в другое (например, GOTO).

Структурное кодирование применяется для программирования отдельных модулей. Вначале на основе положений 1, 2, 3 разрабатывается алгоритм, а затем базовые алгоритмические конструкции заменяются соответствующими конструкциями конкретного языка. Структурный контроль необходим при разработке сложных задач, включающих десятки и сотни модулей. Это новая форма контроля процесса разработки программ, в которой участвует несколько человек. По мере продвижения в разработке задачи периодически определенной группе сотрудников раздаются рабочие материалы по данной задаче. Они знакомятся с ними, а затем на совместном обсуждении выявляются недостатки данного этапа, которые разработчик программы должен в ближайшее время устранить.

Метод – это блок кода, содержащий ряд инструкций. Программа инициирует выполнение инструкций, вызывая метод и указывая все аргументы, необходимые для этого метода. В C# все инструкции выполняются в контексте метода. Метод Main является точкой входа для каждого приложения C# и вызывается общезыковой средой выполнения (CLR) при запуске программы.

### **Сигнатуры методов**

Методы объявляются с помощью ограничений class, record или struct, для которых указываются следующие данные.

- Уровень доступа (необязательно), например public или private. Значение по умолчанию – private.
- Необязательные модификаторы, например abstract или sealed.
- Возвращаемое значение или void, если у метода его нет.
- Имя метода.

- Любые параметры методов. Параметры метода заключаются в скобки и разделяются запятыми. Пустые скобки указывают, что параметры методу не требуются.

Вместе все эти части формируют сигнатуру метода.

### **Доступ к методу**

Вызов метода в объекте аналогичен доступу к полю. После имени объекта добавьте точку, имя метода и круглые скобки. Аргументы перечисляются в этих скобках и разделяются запятыми.

### **Параметры и аргументы метода**

Определение метода задает имена и типы всех необходимых параметров. Когда вызывающий код вызывает метод, он предоставляет конкретные значения, называемые аргументами, для каждого параметра. Аргументы должны быть совместимы с типом параметра, но имя аргумента (если есть), используемое в вызывающем коде, не обязательно должно совпадать с именем параметра, указанным в методе.

### **Возвращаемые значения**

Методы могут возвращать значение вызывающему объекту. Если тип возврата, указываемый перед именем метода, не `void`, этот метод может возвращать значение с помощью оператора `return`. Инструкция с ключевым словом `return`, за которым следует значение, соответствующее типу возврата, будет возвращать это значение объекту, вызвавшему метод.

Значение можно вернуть вызывающему объекту по значению или по ссылке. Значения возвращаются вызывающему объекту по ссылке, если ключевое слово `ref` используется в сигнатуре метода и указывается после каждого ключевого слова `return`. Ключевое слово `return` также останавливает выполнение метода. Если тип возврата – `void`, инструкцию `return` без значения по-прежнему можно использовать для завершения выполнения метода. Без ключевого слова `return` этот метод будет останавливать выполнение при достижении конца блока кода. Методы с типом возврата, отличным от `void`, должны использовать ключевое слово `return` для возврата значения.



## **КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. Что такое подпрограмма?
2. Какие параметры называются фактическими?
3. Какие параметры называются формальными?
4. Как связаны между собой формальные и фактические параметры?
5. Какие переменные называются глобальными?
6. Какие переменные называются локальными?

## **САМОСТОЯТЕЛЬНАЯ РАБОТА №6**

### **Раздел 6. Решение прикладных задач**

#### **Тема 6.1. Постановка задачи**

Содержание темы:

1. Анализ, формальная постановка и выбор метода решения задачи.
2. Разработка алгоритма решения задачи.

Тема 6.2. Проектирование объектно-ориентированного приложения

Содержание темы:

1. Создание интерфейса приложения.
2. Разработка программных модулей.
3. Тестирование, отладка приложения.

## **МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ**

В процессе изучения данного раздела студент должен усвоить, что при создании программного обеспечения необходимо пройти несколько этапов:

- **Постановка задачи.** На этом этапе осуществляется формулирование цели решения задачи, описывается входная и выходная информация. Входная информация по задаче – данные, поступающие на вход задачи и используемые для её решения. Выходная информация может быть представлена в виде документов, кадров на экране монитора, информации в базе данных, выходного сигнала устройству управления. Постановка задачи разрабатывается организацией, разработчиком программной продукции, на основании технического задания совместно с заказчиком. Главный исполнитель – это разработчик.

- **Математическое описание задачи.** На этом этапе выражаются существующие соотношения между исследуемыми величинами посредством математических формул, указываются допущения и ограничения.

Математическая постановка включает в себя:

- 1) выбор, обоснование и описание модели исследуемого объекта;
- 2) определение цели исследования;

3) выбор и описание методов решения поставленных задач (достижения цели исследования).

Для построения математической модели на основании словесной постановки задачи выбираются переменные, подлежащие определению, записываются ограничения и выявляются связи между переменными.

В общем случае, математические модели исследуемых объектов могут быть числовыми (с конкретными числовыми значениями характеристик), логическими (в основе логические выражения) и графическими (графики, диаграммы, рисунки, которые легко преобразуются в математические формулы). Разрабатываемая математическая модель может отражать внутреннюю структуру объекта (отношения между составляющими этот объект элементами) или функционирование объекта (зависимости между воздействиями на объект и его состояниями).

При построении математической модели может преследоваться цель определения такого состояния объекта, которое является наилучшим или допустимым в каком-либо смысле. Модель может быть предназначена для объяснения наблюдаемых фактов или прогноза поведения объекта.

Примерная структура математической постановки задачи выглядит стандартным образом: дано; требуется получить; решение; выводы.

- Выбор и обоснование метода решения задачи. Выбор методов исследования осуществляется исходя из построенной модели и сформулированной цели. При этом учитывается точность вычислений, время решения задачи на ЭВМ, требуемый объем памяти и т.д.

- Алгоритмизация вычислительного процесса. На этом этапе строится детальный алгоритм программы. При этом процесс обработки данных разбивается на отдельные блоки, устанавливается последовательность выполнения блоков, разрабатывается блок-схема алгоритма программы.

- Проектирование базы данных.

- Написание программного кода. На этом этапе создаётся программный код на заданном языке программирования в соответствии с составленным алгоритмом.

- Отладка программы, поиск и устранение синтаксических и логических ошибок. Отладка – этап разработки компьютерной программы, на котором обнаруживают, локализуют и устраняют ошибки.

Чтобы понять, где возникла ошибка, приходится:

- 1) узнавать текущие значения переменных;
- 2) выяснять, по какому пути выполнялась программа.

Существуют две взаимодополняющие технологии отладки:

- 1) Использование отладчиков – программ, которые включают в себя пользовательский интерфейс для пошагового выполнения программы: оператор за оператором, функция за функцией, с остановками на некоторых строках исходного кода или при достижении определённого условия;

- 2) Вывод текущего состояния программы с помощью расположенных в критических точках программы операторов вывода – на экран, принтер, громкоговоритель или в файл. Вывод отладочных сведений в файл называется журналированием.

То есть, при написании программного кода проще сделать так, чтобы отладка нужна была как можно реже. Для этого применяются следующие методы:

- статический анализ кода. На этой фазе программа сканер ищет последовательности в исходном тексте, соответствующие небезопасным вызовам функций и т. Д. Фактически идет сканирование исходного текста программы на основе специальной базы правил, которая содержит описание небезопасных образцов кода;
- фаззинг. Это процесс подачи на вход программы случайных или некорректных данных и анализ реакции программы;
- Reverse engineering (Обратная инженерия). Этот случай возникает, когда независимые исследователи ищут уязвимости и недокументированные возможности программы.

## **КОНТРОЛЬНЫЕ ВОПРОСЫ**

1. В чем заключается анализ, формальная постановка и выбор метода решения задачи?
2. Опишите этапы постановки задачи
3. Опишите принципы разработки алгоритмов решения задачи.
4. Опишите процесс программирования модулей, тестирования и отладки программы
5. Опишите цели создания прикладного программного обеспечения.
6. Опишите любую прикладную задачу в области организации перевозок с помощью математических формул.
7. Опишите процесс создания базы данных в Excel, Access.
8. Каким способом лучше описать алгоритм работы подвижного состава на линии?
9. Опишите назначение блоков, входящих в блок-схему алгоритма прикладной программы.
10. Проанализируйте результаты, полученные в результате решения задачи на ЭВМ.

### Литература

1. Семакин, И. Г. Основы алгоритмизации и программирования : учебник для образовательных организаций, реализующих программы среднего профессионального образования по специальностям "Информационные системы и программирование", "Сетевое и системное администрирование", "Обеспечение информационной безопасности автоматизированных систем", "Обеспечение информационной / И. Г. Семакин, А. П. Шестаков ; И. Г. Семакин, А. П. Шестаков. – 4-е изд., стер. – Москва : Академия, 2021. – с. 304.
2. Трофимов, В. В. Основы алгоритмизации и программирования.: учебник для СПО / Трофимов В. В., Павловская Т. А. ; Под ред. Трофимова В.В.. – Москва : Юрайт, 2023. – 137 с.
3. Голицына, О. Л. Информационные системы и технологии : Учебное пособие / О. Л. Голицына, Н. В. Попов И. И. Максимов. – Москва : НИЦ ИНФРА-М, 2023. – 400 с.
4. Федотова, Е. Л. Информационные технологии в профессиональной деятельности : Учебное пособие / Е. Л. Федотова. – Москва : НИЦ ИНФРА-М, 2024. – 367 с.